Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX

Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX

Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto and Aiko Pras

***Abstract*—Flow monitoring has become a prevalent method for monitoring traffic in high-speed networks. By focusing on the analysis of flows, rather than individual packets, it is often said to be more scalable than traditional packet-based traffic analysis. Flow monitoring embraces the complete chain of packet observation, flow export using protocols such as NetFlow and IPFIX, data collection, and data analysis. In contrast to what is often assumed, all stages of flow monitoring are closely intertwined. Each of these stages therefore has to be thoroughly understood, before being able to perform sound flow measurements. Otherwise, flow data artifacts and data loss can be the consequence, potentially without being observed.**

This paper is the first of its kind to provide an integrated tutorial on all stages of a flow monitoring setup. As shown throughout this paper, flow monitoring has evolved from the early nineties into a powerful tool, and additional functionality will certainly be added in the future. We show, for example, how the previously opposing approaches of Deep Packet Inspection and flow monitoring have been united into novel monitoring approaches.

***Index Terms*—Flow export, network monitoring, Internet measurements, NetFlow, IPFIX**

I. Introduction

NETWORK monitoring approaches have been proposed and developed throughout the years, each of them serving a different purpose. They can generally be classified into two categories: active and passive. Active approaches, such as implemented by tools like Ping and Traceroute, inject traffic into a network to perform different types of measurements. Passive approaches observe existing traffic as it passes by a measurement point and therefore observe traffic generated by users. One passive monitoring approach is packet capture. This method generally provides most insight into the network traffic, as complete packets can be captured and further analyzed. However, in high-speed networks with line rates of up to 100 Gbps, packet capture requires expensive hardware and substantial infrastructure for storage and analysis.

Another passive network monitoring approach that is more scalable for use in high-speed networks is flow export, in which packets are aggregated into flows and exported for storage and analysis. A flow is defined in [1] as "a set of IP packets passing an observation point in the network during a certain time interval, such that all packets belonging to a particular flow have a set of common properties". These common properties may include packet header fields, such as source and destination IP addresses and port numbers, packet contents, and meta-information. Initial works on flow export date back to the nineties and became the basis for modern protocols, such as NetFlow and IP Flow Information eXport (IPFIX) [2].

In addition to their suitability for use in high-speed networks, flow export protocols and technologies provide several other advantages compared to regular packet capture. First, they are widely deployed, mainly due to their integration into high-end packet forwarding devices, such as routers, switches and firewalls. For example, a recent survey among both commercial and research network operators has shown that 70% of the participants have devices that support flow export [3]. As such, no additional capturing devices are needed, which makes flow monitoring less costly than regular packet capture. Second, flow export is well understood, since it is widely used for security analysis, capacity planning, accounting, and profiling, among others. It is also frequently used to comply to data retention laws. For example, communication providers in Europe are enforced to retain connection data, such as provided by flow export, for a period of between six months and two years "for the purpose of the investigation, detection and prosecution of serious crime" [4], [5]. Third, significant data reduction can be achieved – in the order of 1/2000 of the original volume, as shown in this paper – since packets are aggregated after they have been captured. Fourth, flow export is usually less privacy-sensitive than packet export, since traditionally only packet headers are considered. However, since researchers, vendors and standardization organizations are working on the inclusion of application information in flow data, the advantage of performing flow export in terms of privacy is fading.

Despite the fact that flow export, as compared to packet-level alternatives, significantly reduces the amount of data to be analyzed, the size of flow data repositories can still easily exceed tens of terabytes. This high volume, combined with the

Rick Hofstede, Anna Sperotto and Aiko Pras are with the University of Twente, Centre for Telematics and Information Technology (CTIT), P.O. Box 217, 7500 AE Enschede, The Netherlands (email: {r.j.hofstede, a.sperotto, a.pras}@utwente.nl).

Pavel Čeleda is with the Masaryk University, Institute of Computer Science, Botanická 68a, 602 00 Brno, Czech Republic (email: celeda@ics.muni.cz).

Brian Trammell is with ETH Zürich, Communication Systems Group, Gloriastrasse 35, 8092 Zürich, Switzerland (email: trammell@tik.ee.ethz.ch).

Idilio Drago is with the Politecnico di Torino, Department of Electronics and Telecommunications, Corso Duca Degli Abruzzi 24, 10129, Torino, Italy (email: idilio.drago@polito.it).

Ramin Sadre is with the Aalborg University, Department of Computer Science, Distributed and Embedded Systems, Selma Lagerlöfs Vej 300, 9220 Aalborg, Denmark (email: rsadre@cs.aau.dk).

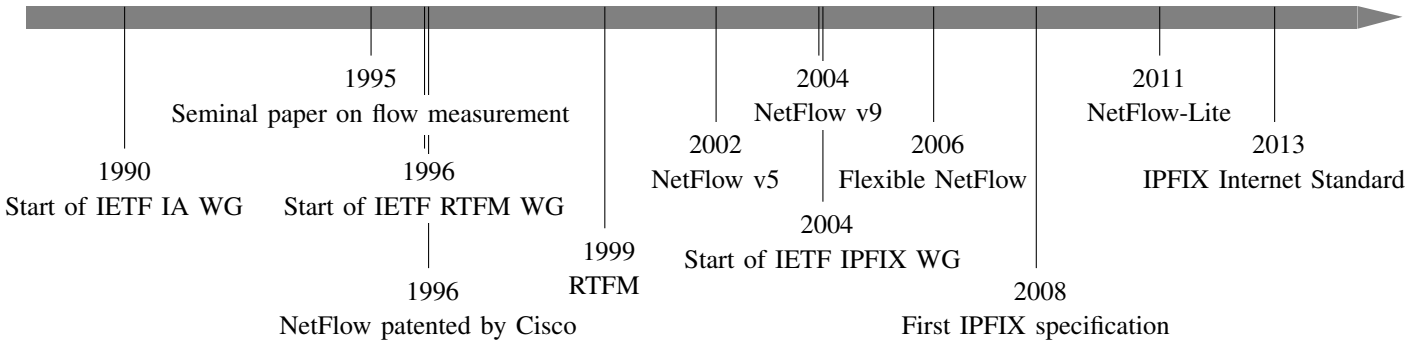


Fig. 1. Evolution of flow export technologies and protocols.

high speeds at which the data is processed, and the increasing types of information that can be exported, make that flow data can be considered a form of "Big Data". Capturing, collecting and analyzing data is therefore a challenging task, which is the main incentive for this paper.

A. Objective

Many papers, specifications and other documents on NetFlow and IPFIX have been written over the years. They usually consider the proper operation of flow export protocols and technologies, and the correctness of the exported data as a given, while barely discussing how to perform sound and accurate flow measurements. This paper focusses especially on that. The objective of this tutorial is to provide a clear understanding of flow export and all stages in a typical flow monitoring setup, covering the complete spectrum from packet capture to data analysis. After reading this tutorial, the reader should be able to decide which choices to make when setting up a flow monitoring infrastructure. The reader is assumed to be familiar with basic networking protocols, but not necessarily with flow export and network measurements.

B. Approach

We have used several approaches for collecting the information contained in this paper. First, we surveyed the literature where relevant and applicable. As a complement to this survey, we have included information based on our own experience. This experience has been gained in a variety of ways: research in the area of flow export, involvement in the standardization of IPFIX, operational experience, talks with network operators, and experience in developing both hardware- and software-based flow exporters. Finally, we have included measurements to illustrate and provide more examples and insights into the presented concepts. These measurements are based on a packet trace consisting of one day of network traffic between the campus network of the University of Twente (UT), and the Dutch national research and education network SURFnet, accounting for 2.1 TB of data. Various sections throughout this paper will refer to these measurements.

C. Organization

The remainder of this paper is organized as follows. Section II shortly outlines the history of flow export protocols, and puts flow export in a broader context by comparing it to related technologies. An overview of a typical flow monitoring architecture and the most important concepts is presented in Section III, which is also the basis for Section IV–VII. In these sections, each of the stages of the architecture is described in detail. In Section VIII, we describe several lessons learned that are most important when setting up flow monitoring, based on our experience over the past decade. We close this paper in Section IX, where we draw our conclusions. A list of acronyms is provided as Appendix.

D. How to Read this Paper?

Although this tutorial targets a wide audience of both experts in the field of flow monitoring and people unfamiliar with the subject, we believe that some sections are more relevant to certain audiences than others. Section II and III are intended for all readers, as they provide a general background on flow monitoring. Researchers interested in creating or revising an existing flow monitoring setup for the sake of network measurements are encouraged to study all contents of Section IV–VII as well. Network operators that use an existing packet forwarding infrastructure for flow monitoring, can skip the material on packet capture in Section IV, as packet capture functionality is typically an integral part of such networking devices. Readers new to flow monitoring are advised to focus on Section V besides Section II and III. Finally, we believe that Section VIII provides useful insights to all readers.

II. History & Context

In this section, we discuss both the history of flow monitoring and present flow monitoring in a broader context by comparing it to related technologies. The chronological order of the main historic events in this area is shown in Fig. 1 and will be covered in Section II-A. A comparison with related technologies and approaches is provided in Section II-B.

A. History

The published origins of flow export date back to 1991, when the aggregation of packets into flows by means of packet header information was described in [6]. This was done as part of the Internet Accounting (IA) Working Group (WG) of the Internet Engineering Task Force (IETF). This WG

concluded in 1993, mainly due to lack of vendor interest. Also the then-common belief that the Internet should be free, meaning that no traffic capturing should take place that could potentially lead to accounting, monitoring, etc., was a reason for concluding the WG. In 1995, interest in exporting flow data for traffic analysis was revived by [7], which presented a methodology for profiling traffic flows on the Internet based on packet aggregation. One year later, in 1996, the new IETF Realtime Traffic Flow Measurement (RTFM) WG was chartered with the objectives of investigating issues in traffic measurement data and devices, producing an improved traffic flow model, and developing an architecture for improved flow measurements. This WG revised the Internet Accounting architecture and, in 1999, published a generic framework for flow measurement, named RTFM Traffic Measurement System, with more flexibility in terms of flow definitions and support for bidirectional flows [8]. In late 2000, having completed its charter, the RTFM WG was concluded. Again, due to vendors' lack of interest, no flow export standard resulted.

In parallel to RTFM, Cisco worked on its flow export technology named NetFlow, which finds its origin in switching. In flow-based switching, flow information is maintained in a *flow cache* and forwarding decisions are only made in the control plane of a networking device for the first packet of a flow. Subsequent packets are then switched exclusively in the data plane [9]. The value of the information available in the flow cache was only a secondary discovery [10] and the next step to export this information proved to be relatively small. NetFlow was patented by Cisco in 1996. The first version to see wide adoption was NetFlow v5 [11], which became available to the public around 2002. Although Cisco never published any official documentation on the protocol, the widespread use was in part result of Cisco making the corresponding data format freely available [2]. NetFlow v5 was obsoleted by the more flexible NetFlow v9, the state of which as of 2004 is described in [12]. NetFlow v9 introduced support for adaptable data formats through templates, as well as IPv6, Virtual Local Area Networks (VLANs) and Multiprotocol Label Switching (MPLS), among other features. Several vendors besides Cisco provide flow export technology alike NetFlow v9 (e.g., Juniper's J-Flow), which are mostly compatible with NetFlow v9. The flexibility in representation enabled by NetFlow v9 made other recent advances possible, such as more flexibility in terms of flow definitions. Cisco provides this functionality by means of its Flexible NetFlow technology [13]. Later, in 2011, Cisco presented NetFlow-Lite, a technology based on Flexible NetFlow that uses an external packet aggregation machine to facilitate flow export on packet forwarding devices without flow export capabilities, such as datacenter switches [14].

Partly in parallel to the NetFlow development, the IETF decided in 2004 to standardize a flow export protocol, and chartered the IP Flow Information Export (IPFIX) WG [15]. This WG first defined a set of requirements [16] and evaluated several candidate protocols. As part of this evaluation, NetFlow v9 was selected as the basis of the new IPFIX Protocol [17]. However, IPFIX is not merely "the standard version of NetFlow v9" [18], as it supports many new features. The first specifications were finalized in early 2008, four years after the IPFIX WG was chartered. These specifications were the basis of what has become the IPFIX Internet Standard [1] in late 2013. A short history on flow export and details on development and deployment of IPFIX are provided in [2].

Note that the term *NetFlow* itself is heavily overloaded in literature. It refers to multiple different versions of a Cisco-proprietary flow export protocol, of which there are also third-party compatible implementations. It refers as well to a flow export technology, consisting of a set of packet capture and flow metering implementations that use these export protocols. For this reason, we use the term *flow export* in this paper to address exporting in general, without reference to a particular export protocol. As such, the term *NetFlow* is solely used for referencing the Cisco export protocol.

B. Related Technologies & Approaches

There are several related technologies with *flow* in the name that do not solve exactly the same problems as flow export. One is sFlow [19], an industry standard integrated into many packet forwarding devices for sampling packets and interface counters. Its capabilities for exporting packet data chunks and interface counters are not typical features of flow export technologies. Another difference is that flow export technologies also support 1:1 packet sampling, i.e., considering every packet for data export, which is not supported by sFlow. From an architectural perspective, which will be discussed in Section III for NetFlow and IPFIX, sFlow is however very similar to flow export technologies. Given its packet-oriented nature, sFlow is closer related to packet sampling techniques, such as the Packet SAMPling (PSAMP) standard [20] proposed by the IETF, than to a flow export technology. Given that this paper is about flow export, we do not consider sFlow.

Another related technology, which is rapidly gaining attention in academia and network operations, is OpenFlow [21]. Being one particular technology for Software-Defined Networking (SDN), it separates the control plane and data plane of networking devices [22]. OpenFlow should therefore be considered a flow-based configuration technology for packet forwarding devices, instead of a flow export technology. Although it was not specifically developed for the sake of data export and network monitoring, as is the case for flow export technologies, flow-level information available within the OpenFlow control plane (e.g., packet and byte counters) was recently used for performing network measurements [23]. Tutorials on OpenFlow are provided in [24], [25].

A data analysis approach that is often related to flow export is Deep Packet Inspection (DPI), which refers to the process of *analyzing* packet payloads. Two striking differences can be identified between DPI and flow export. First, flow export traditionally only considers packet headers, and is therefore considered less privacy-sensitive than DPI and packet export. Second, flow export is based on the aggregation of packets (into flows), while DPI and packet export are typically considering individual packets. Although seemingly opposing, we show throughout this paper how DPI and flow export are increasingly united for increased visibility in networks.

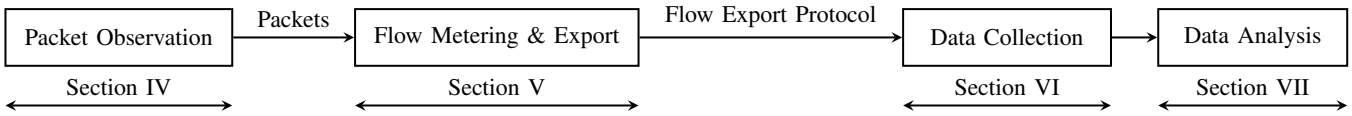


Fig. 2. Architecture of a typical flow monitoring setup.

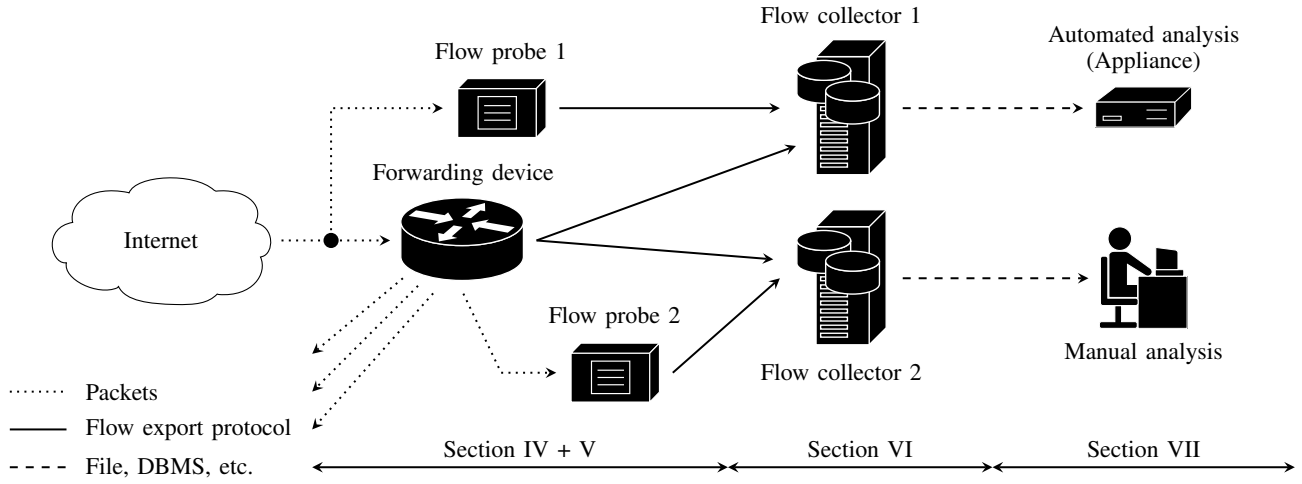


Fig. 3. Various flow monitoring setups.

III. Flow Monitoring Architecture

The architecture of typical flow monitoring setups consists of several stages, each of which is shown in Fig. 2. The first stage is Packet Observation, in which packets are captured from an *Observation Point* and pre-processed. Observation Points can be line cards or interfaces of packet forwarding devices, for example. We discuss the Packet Observation stage in Section IV.

The second stage is Flow Metering & Export, which consists of both a *Metering Process* and an *Exporting Process*. Within the Metering Process, packets are aggregated into flows, which are defined as "sets of IP packets passing an observation point in the network during a certain time interval, such that all packets belonging to a particular flow have a set of common properties" [1]. After a flow is considered to have terminated, a flow record is exported by the Exporting Process, meaning that the record is placed in a datagram of the deployed flow export protocol. Flow records are defined in [1] as "information about a specific flow that was observed at an observation point", which may include both characteristic properties of a flow (e.g., IP addresses and port numbers) and measured properties (e.g., packet and byte counters). They can be imagined as records or rows in a typical database, with one column per property. The *Metering* and *Exporting* processes are in practice closely related. We therefore discuss these processes together in Section V.

The third stage is Data Collection, which is described in Section VI. Its main tasks are the reception, storage and pre-processing of flow data generated by the previous stage. Common pre-processing operations include aggregation, filtering, data compression, and summary generation.

The final stage is Data Analysis, which is discussed in Section VII. In research deployments, data analysis is often of an exploratory nature (i.e., manual analysis), while in operational environments, the analysis functions are often integrated into the Data Collection stage (i.e., both manual and automated). Common analysis functions include correlation and aggregation; traffic profiling, classification, and characterization; anomaly and intrusion detection; and search of archival data for forensic or other research purposes.

Note that the entities within the presented architecture are conceptual, and may be combined or separated in various ways, as we exemplify in Fig. 3. We will highlight two important differences. First and most important, the Packet Observation and Flow Metering & Export stages are often combined in a single device, commonly referred to as *Flow Export Device* or *flow exporter*. When a flow exporter is a dedicated device, we refer to it as *flow probe*. Both situations are shown in Fig. 3. We know however from our own experience that the IPFIX architecture [26] was developed with flow export from packet forwarding devices in mind. In this arrangement, packets are read directly from a monitored link or received via the forwarding mechanisms of a packet forwarding device. However, especially in research environments where trace data is analyzed, packet capture may occur on a completely separate device, and as such should not be considered an integral part of the Flow Metering & Export stage. This is why we consider the Packet Observation and Flow Metering & Export stages in this work to be separate. A second difference with what is shown in Fig. 2, is that multiple flow exporters can export flows to multiple devices for storage and pre-processing, commonly referred to as *flow collectors*. After pre-processing, flow data is available for analysis, which can be both automated (e.g., by means of an *appliance*) or manual.

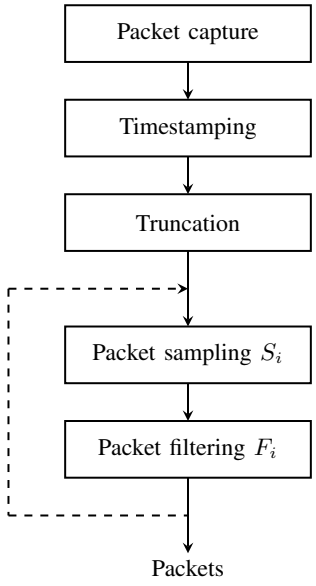


Fig. 4. Architecture of the Packet Observation stage.

IV. Packet Observation

Packet observation is the process of capturing packets from the line and pre-processing them for further use. It is therefore key to flow monitoring. In this section, we cover all aspects of the Packet Observation stage, starting by presenting its architecture in Section IV-A. Understanding this architecture is however not enough for making sound packet captures; also the installation and configuration of the capture equipment is crucial. This is explained in Section IV-B. Closely related to that are special packet capture technologies that help to increase the performance of capture equipment, which is surveyed in Section IV-C. Finally, in Section IV-D, we discuss one particular aspect of the Packet Observation stage in detail that is widely used in flow monitoring setups: packet sampling & filtering.

A. Architecture

A generic architecture of the Packet Observation stage is shown in Fig. 4. Before any packet pre-processing can be performed, packets must be read from the line. This step, packet capture, is the first in the architecture and typically carried out by a Network Interface Card (NIC). Before packets are stored in on-card reception buffers and later moved to the receiving host's memory, they have to pass several checks when they enter the card, such as checksum error checks.

The second step is timestamping. Accurate packet timestamps are essential for many processing functions and analysis applications. For example, when packets from different observation points have to be merged into a single dataset, they will be ordered based on their timestamps. Timestamping performed in hardware upon packet arrival avoids delays as a consequence of forwarding latencies to software, resulting in an accuracy of up to 100 nanoseconds in the case of the IEEE 1588 protocol, or even better. Unfortunately, hardware-based timestamping is typically only available on special NICs using Field Programmable Gate Arrays (FPGAs), and most commodity cards perform timestamping in software. However, software-based clock synchronization by means of the Network Time Protocol (NTP) or the Simple Network Time Protocol (SNTP) usually provides an accuracy in the order of 100 microseconds. For further reading, we recommend the overviews on time synchronization methods in [27], [28].

Both packet capture and timestamping are performed for all packets under any condition. All subsequent steps shown in Fig. 4, are optional. The first of them is packet truncation, which selects only those bytes of a packet that fit into a pre-configured *snapshot length*. This reduces the amount of data received and processed by a capture application, and therefore also the number of computation cycles, bus bandwidth and memory used to process the network traffic. For example, flow exporters traditionally only rely on packet header fields and ignore packet payloads.

The last step of the Packet Observation stage is packet sampling and filtering [29]. Capture applications may define sampling and filtering rules so that only certain packets are selected for measurement. The motivation for sampling is to select a packet subset, while still being able to estimate properties of the full packet stream. The motivation for filtering is to remove all packets that are not of interest. Packet sampling & filtering will be discussed in detail in Section IV-D.

B. Installation & Configuration

In this subsection, we describe how packet captures should be made in wired, wireless, and virtual networks, and how the involved devices should be installed and configured. Most packet captures are made in wired networks, but can also be made in wireless networks. Due to the popularity of virtual environments, packet captures in virtual networks are also becoming more common.

Most network traffic captures are made in wired networks, which can range from Local Area Networks (LANs) to backbone networks. This is mainly due to their high throughput and low external interference. Packet capture devices can be positioned in-line and in mirroring mode, which may have a significant impact on capture and network operation:

- *In-line mode* – The capture device is directly connected to the monitored link between two hosts. This can be achieved by installing additional hardware, such as bridging hosts or network taps [30]. Network taps[1] are designed to duplicate all traffic passing through the tap and provide a connection for a dedicated capture device. They use passive splitting (optical fiber networks) or regeneration (electrical copper networks) technology to pass through traffic at line rates without introducing delays or altering data. In addition, they have built-in fail open capability that ensures traffic continuity even if a tap stops working or loses power. Once a tap has been installed, capture devices can be connected or disconnected without affecting the monitored link. In Fig. 3, *Flow probe 1* receives its input traffic by means of a network tap.

[1] Another commonly used name is Test Access Port (TAP).

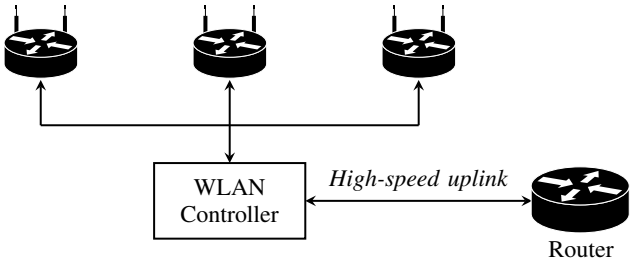


Fig. 5. Packet capture in wireless networks.

- *Mirroring mode* – Most packet forwarding devices can mirror packets from one or more ports to another port, to which a capture device is connected. This is commonly referred to as *port mirroring*, *port monitoring*, or *Switched Port ANalyzer (SPAN) session* [31]. Port mirroring requires a change in the forwarding device's configuration, but does not introduce additional costs as for a network tap. It should be noted that mirroring may introduce delays and jitter, alter the content of the traffic stream, or reorder packets [32]. In addition, care should be taken to select a mirror port with enough bandwidth; given that most captures should cover two traffic directions (full-duplex), the mirror port should have twice the bandwidth of the monitored port, to avoid packet loss. In Fig. 3, *Flow probe 2* receives its input traffic by means of port mirroring.

Packet captures in wireless networks can be made using any device equipped with a wireless NIC, under the condition that the wireless traffic is not encrypted at the link-layer, or the encryption key is known. Wireless NICs can however only capture at a single frequency at a given time. Although some cards support channel hopping, by means of which the card can switch rapidly through all radio channels, there is no guarantee that all packets are captured [33]. In large-scale wireless networks, it is more common to capture all traffic at a Wireless LAN (WLAN) Controller, which controls all individual wireless access points and forwards their traffic to other network segments by means of a high-speed wired interface. This is shown in Fig. 5, where the high-speed uplink suitable with traffic from and to all access points can be captured. Besides having a single point of capture, the advantage is that link-layer encryption of wireless transmission protocols does not play a role anymore and captures can be made as described above for wired networks.

Deployment of packet capture devices in virtual networks is very similar to deployment in wired networks, and is rapidly gaining importance due to the widespread use of virtual machines (VMs). Virtual networks act as wired LANs, but are placed in virtual environments, e.g., to interconnect VMs. We therefore do not consider Virtual Private Networks (VPNs) as virtual networks, as they are typically just overlay networks in physical networks. Virtual networks use virtual switches [34], which support port mirroring and virtual taps. Furthermore, the mirrored traffic can be forwarded to dedicated physical ports and captured outside the virtual environment by a packet capture device.

Key to monitoring traffic is to identify meaningful observation points, ultimately allowing capture devices to gather most information on traffic passing by the observation point. These observation points should preferably be in wired networks. Even in wireless networks one should consider capturing at a WLAN controller to overcome all previously discussed limitations. In addition, deployment of network taps is usually preferred over the use of mirror ports, mainly due to effects on the packet trace of the latter. Port mirroring should only be used if necessary and is particularly useful for ad-hoc deployments and in production networks where no taps have been installed.

C. Technologies

For most operating systems, libraries and Application Programming Interfaces (APIs) for capturing network traffic are available, such as *libpcap* or *libtrace* [35] for Linux and BSD-based operating systems, and *WinPcap* for Windows. These libraries provide both packet capture and filtering engines, and support reading from and writing to offline packet traces. From a technical point of view, they are located on top of the operating system's networking stack.

Since the networking stacks of operating systems are designed for general-purpose networking, packet captures usually suffer from suboptimal performance. The overall capture performance depends on system costs to hand over packets from the NIC to the capture application, via a packet capture library; packets have to traverse several layers, which increase latency and limit the overall performance as they add per-packet processing overhead. Several methods have been proposed to speed up this process [36]:

- Interrupt mitigation and packet throttling (Linux NAPI) reduce performance degradation of the operating system under heavy network loads. Interrupt mitigation decreases the number of interrupts triggered by NICs during times of heavy traffic, as all interrupts convey the same message about a large number of packets waiting for processing. This reduces the system load. Packet throttling is applied when a system is overloaded with packets; packets are already dropped by the NIC, even before they are moved to the software.
- Network stack bypass techniques, such as PF_RING, avoid the per-packet processing overhead caused by the various OS networking layers.
- Memory-map techniques for reducing the cost of copying packets form kernel-space to user-space.

All these methods provide software-based optimizations for making packet captures. To be able to deal with higher packet rates, however, hardware-acceleration cards have been introduced. They use FPGAs to reduce CPU load during packet capture and guarantee packet capture without loss under modest CPU load [37]. Other capabilities of these cards are precise timestamping (with GPS synchronization), traffic filtering, and multi-core traffic distribution by means of multiple receive queues. They use Direct Memory Access (DMA) to receive and transmit packets. In that way, they also

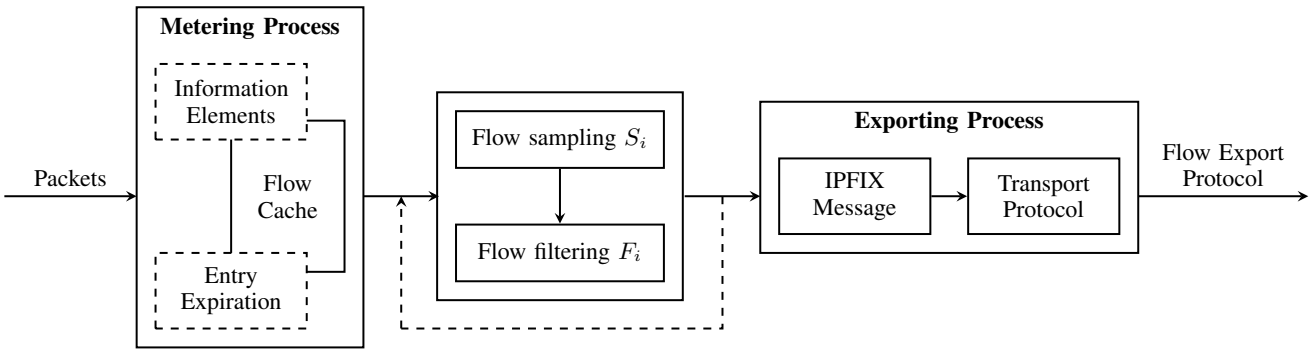


Fig. 6. Architecture of the Flow Metering & Export stage.

address the problem of passing packets efficiently from NICs to the capture application.

Modern commodity NICs provide a cost-effective solution for making high performance packet captures on links with speeds up to 10 Gbps [38]. Features provided by controllers on these NICs (e.g., Intel 82599, Myri-10G Lanai Z8ES) include multiple receive queues by means of *Receive Side Scaling* to distribute packets across multiple CPUs [39], and a DMA engine to off-load CPU processing. To be able to use these features, vendors provide a set of libraries and drivers for fast packet processing, such as Intel DPDK[2] and PF_RING DNA/Libzero[3].

It is important to fully understand the performance of the packet capture process to create and operate reliable monitoring applications. Care should be taken when selecting a packet capture library or system: Most packet capture benchmarks show throughputs for situations without any further processing, which may overestimate the real performance when some form of packet processing is used. An example of such packet processing is flow export, which will be discussed in the subsequent sections. Key to high-performance packet processing is efficient memory management, low-level hardware interaction, and application optimizations.

D. Packet Sampling & Filtering

The goal of packet sampling and filtering is to forward only certain packets to the Flow Metering & Export stage. A combination of several sampling and filtering steps can be adopted to select the packets of interest.

Packet sampling[4] aims at reducing the load of subsequent stages or processes (e.g., the *Metering Process* within the Flow Metering & Export stage) and, consequently, to reduce the consumption of bandwidth, memory and computation cycles. Therefore, sampling should be used whenever it is expected that the number of monitored packets will overload the following stage. The ultimate goal is to turn the uncontrolled loss of packets caused by overload into a controlled one by using sampling.

[2]http://www.dpdk.org/

[3]http://www.ntop.org/products/pf_ring/libzero-for-dna/

[4]See [40] and [41] for an introduction to sampling in the context of network management.

Several sampling strategies are defined in [29], where two major classes of sampling schemes can be distinguished: *Systematic sampling* schemes deterministically decide which packets are selected (for example every Nth packet in a periodic sampling scheme). In contrast, *random sampling* selects packets in accordance to a random process. The main challenge when using sampling is to obtain a representative subset of the relevant packets. In general, random sampling should be preferred over systematic sampling when both are available, because the latter can introduce unwanted correlations in the observed data. For example, a measurement using periodic sampling would be likely biased towards or against periodic traffic. This restriction can be relaxed when it is known in advance that the monitored traffic is highly aggregated, i.e., it comprises of traffic from many different hosts, applications, etc. In such a situation, the influence of the sampling scheme is less noticeable, although its quantitative impact on the resulting flow data depends on the nature of the traffic [42].

Packet sampling obviously entails loss of information. Depending on the employed sampling scheme, some properties of the original packet stream can be easily recovered. For example, if a simple random sampling scheme is used, the total number of packets or bytes can be estimated by multiplying the measured numbers by the inverse of the sampling probability. Reciprocally, it means that sampling with a rate of 1:N results in a reduction of load to the *Metering Process* (in terms of number of packets and bytes to process) by a factor of N. Other characteristics of the original data are affected in a more complex way. For example, longer flows are more likely to be sampled than shorter ones. A simple scaling would yield a biased estimation of the flow length distribution. Methods to estimate sampled flow statistics have been discussed in [42]. Several publications propose new sampling schemes with the goal to mitigate the effects of sampling, for example by automatically adapting the sampling rate according to the traffic load [43].

The role of *packet filtering* is to deterministically "separate all the packets having a certain property from those not having it" [29]. Similar to sampling, filtering can be used to reduce the amount of data to be processed by the subsequent stages. Again, two major classes can be distinguished: With *Property Match Filtering*, a packet is selected if specific fields within the

TABLE I
COMMON IPFIX INFORMATION ELEMENTS [44]

ID	Name	Description
152	flowStartMilliseconds	Timestamp of the flow's first packet.
153	flowEndMilliseconds	Timestamp of the flow's last packet.
8	sourceIPv4Address	IPv4 source address in the packet header.
12	destinationIPv4Address	IPv4 destination address in the packet header.
7	sourceTransportPort	Source port in the transport header.
11	destinationTransportPort	Destination port in the transport header.
4	protocolIdentifier	IP protocol number in the packet header.
2	packetDeltaCount	Number of packets for the flow.
1	octetDeltaCount	Number of octets for the flow.

packet (and/or the router state) are equal to a specified value or inside a specified value range. Typically, such filters are used to limit packet capturing to specific IP addresses, applications, etc. *Hash-Based Filtering* applies a hash function to the packet content or some portion of it, and compares the result to a specified value or value range. Hash-Based Filtering can be used to efficiently select packets with common properties or, if the hash function is applied to a large portion of the packet content, to select packets quasi-randomly.

V. FLOW METERING & EXPORT

The Flow Metering & Export stage is where packets are aggregated into flows and flow records are exported, which makes it key to any flow monitoring system. Its architecture is shown in Fig. 6. The packet aggregation is performed within the *Metering Process*, based on Information Elements that define the layout of a flow. Information Elements are discussed in Section V-A. After aggregation, an entry per flow is stored in a *flow cache*, explained in Section V-B, until a flow is considered to have terminated and the entry is expired. Expiration of flow cache entries is discussed in Section V-C. After one or more optional flow-based sampling and filtering functions, which are discussed in Section V-D, flow records have to be encapsulated in messages. This is where IPFIX comes in, which is defined in [18] as "a unidirectional, transport-independent protocol with flexible data representation". IPFIX message structures and types are discussed in Section V-E. Furthermore, a transport protocol has to be selected, which is discussed in Section V-F. Finally, we provide an extensive analysis of open-source and commercial flow metering and export implementations in Section V-G.

A. Information Elements

Fields that can be exported in IPFIX flow records are named *Information Elements* (IEs). The Internet Assigned Numbers Authority (IANA) maintains a standard list of IEs as the IPFIX Information Element registry [44]. Use of the IANA registry for common IEs is key to cross-vendor operability in IPFIX. Besides IANA IEs, enterprise-specific IEs can be

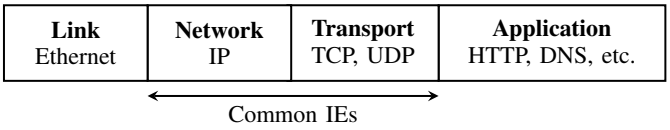


Fig. 7. Network layers considered for IEs.

defined, allowing for new fields to be specified for a particular application without any alterations to IANA's registry. IEs have a name, numeric ID, description, type, length (fixed or variable), and status (i.e., *current* and *deprecated*), together with an enterprise ID in the case of enterprise-specific IEs [45].

A subset of IEs defined in [44] is shown in Table I, which is often considered the smallest set of IEs for describing a flow. These IEs are for transport-layer and network-layer fields, and supported by most flow exporters. However, in contrast to what the name "IP flow information eXport" (IPFIX) suggests, IEs can be defined for any layer, ranging from the link-layer (L2) up to and including the application-layer (L7), as shown in Fig. 7. For example, IEs have been defined for Ethernet [46], such as `sourceMacAddress` (ID 56) and `vlanID` (ID 58). We refer to the support for application-layer IEs as *application awareness*. In other words, flow exporters with application awareness combine DPI with traditional flow export.

There are also other IEs that are different from the default transport- and network-layer IEs shown in Table I in terms of type and semantic. For example, since many IEs are identical to what can be retrieved using widely used Simple Network Management Protocol (SNMP) Management Information Base (MIB) variables, such as `interfaceName` (ID 82), a current standardization effort is working to define a method for exporting SNMP MIB variables using IPFIX [47]. This avoids the repetitive definition of IEs in the IANA registry. Another example are IEs for exporting octet sequences, such as `ipPayloadPacketSection` (ID 314), which can be useful for exporting sampled packet chunks.

Guidelines on the definition of globally unique IEs are provided in [48], which are intended for both those defining new IEs and reviewing the specifications of new IEs. Before defining a new IE, one should be sure to define an IE that 1) is unique within the IANA IE registry, 2) is self-contained, and 3) represents nonproprietary information. After definition, the IE specification should be sent to IANA, after which the request for approval is evaluated by a group of experts, named "IE-Doctors". Upon approval, IANA is requested to apply the necessary changes to the IE registry. The same process applies to requests for IE deprecation.

The configuration of Metering Processes in terms of IEs is not standardized and varies from exporter to exporter. However, flow collectors are always instructed by flow exporters by means of *templates*, which are used to describe which IEs are used for which flow. This approach is also used by NetFlow v9, although it is not compatible with IPFIX, because of the different message formats used by the two protocols. NetFlow v5 does not provide template support and is therefore fixed to its initial specification. This considerably limits the applicability of NetFlow v5, since no protocol evolution is pos-

sible. NetFlow v5 cannot be used for monitoring IPv6 traffic, for example. It is however often suggested that NetFlow v5 is the most widely deployed flow export protocol and therefore still a relevant source of flow information [49], [50].

In addition to what has been described before, several more advanced mechanisms with respect to IEs have been defined: variable-length encoding and structured data. Variable-length encoding can be used for IEs with a variable length in IPFIX, despite of IPFIX' template mechanism being optimized for fixed-length IEs [1]. As such, longer IEs can be transferred efficiently since no bytes are wasted due to a fixed-size reservation. Structured data in IPFIX [51] is useful for transferring a list of the same IE, by encapsulating it in a single field. A clear use-case for this are MPLS labels; since every packet can carry a stacked set of such labels, one traditionally has to define a dedicated IE for every label position, e.g., `mplsTopLabelStackSection`, `mplsTopLabelStackSection2`, etc. With structured data, an MPLS label stack can be encoded using a single IE.

B. Flow Caches

Flow caches are tables within a Metering Process that store information regarding all active network traffic flows. These caches have entries that are composed of IEs, each of which can be either a key or non-key field. The set of key fields, commonly referred to as the *flow key*, is used to determine whether a packet belongs to a flow already in the cache or to a new flow, and therefore defines which packets are grouped into which flow. In other words, the flow key defines the properties that distinguish flows. Incoming packets are hashed based on the flow key and matched against existing flow cache entries. A match triggers a flow cache update where packet and byte counters are tallied. Packets not matching any existing entry in the flow cache are used to create new entries. Commonly used key fields are source and destination IP addresses and port numbers. Non-key fields are used for collecting flow characteristics, such as packet and byte counters.

Given that source and destination IP addresses are normally part of the flow key, flows are usually unidirectional. In situations where both forward and reverse flows (between a source/destination pair) are important, bidirectional flows [52] may be considered. Bidirectional flow records have counters for both directions, and a special IE (`biflowDirection`, ID 239) to indicate a flow's initiator and responder. Since source and destination IP addresses are still part of the flow key in a setup for bidirectional flows, special flow cache support is needed for identifying matching forward and reverse flows.

Several parameters should be considered when selecting or configuring a flow cache for a particular deployment, such as the cache layout, type and size. The flow cache layout should match the selection of key and non-key fields, as these are the IEs accounted for each flow. Given that there are many types of IEs available, flow cache layouts should be able to cope with this flexibility. For example, application information in flow records is becoming more and more important, which can be concluded both from the fact that IEs are being registered for applications in IANA's IE registry, as well as flow exporters with application identification support are being developed. Flow caches, thus, should support flexible flow definitions per application.

Flow caches can also differ from each other in terms of type. For example, IPFIX defines flows that consist of a single packet, commonly referred to as *single-packet flows*[5] [26]. A regular flow cache typically cannot be used for single-packet flows, as the cache management (e.g., the process that determines which flow has terminated) of such caches is often not fast enough. To overcome this problem, some vendors implement dedicated caches for such flows, sometimes referred to as *immediate cache* [53]. An example use case for single-packet flows and immediate caches is a configuration with a very low packet sampling rate, such as 1:2048, where it is expected that no more than one packet is sampled per flow. In those situations, one can avoid resource-intensive cache management by using an *immediate cache*. Besides caches for single-packet flows, it is possible to use caches from which flow entries cannot expire, but are periodically exported, named *permanent cache* [53]. These caches can be used for simple flow accounting, as they do not require a flow collector for collecting flow records; as flow cache entries are never expired, packet and byte counters are never reset upon expiration and therefore represent the flow state since the Metering Process has started.

The size of flow caches depends on the memory available in a flow exporter and should be configured/selected based on the expected number of flows, the selected key and non-key fields, and expiration policies. Given that expiration policies have the strongest impact on the required flow cache size, we discuss them in the next subsection.

C. Flow Cache Entry Expiration

Cache entries are maintained in the flow cache until the corresponding flows are considered to have terminated, after which the entries are expired. These entries are usually expired by the Metering Process according to given timeout parameters or when specific events have occurred. IPFIX, however, does not mandate precisely when flow entries need to be expired and flow records exported. Instead, it provides the following reasons as guidelines on how Metering Processes should expire flow cache entries [26]:

- *Active timeout* – The flow has been active for a specified period of time. Therefore, the active timeout helps to report the activity of long-lived flows periodically. Typical timeout values range from 120 seconds to 30 minutes. Note that cache entries expired using the active timeout are not removed from the cache; counters are reset, and start and end times are updated.
- *Idle timeout* – No packets belonging to a flow have been observed for a specified period of time. Typical timeout values range from 15 seconds to 5 minutes.
- *Resource constraints* – Special heuristics, such as the automatic reduction of timeout parameters at run-time,

[5]In terms of expiration, which is discussed in Section V-C, these flows are said to have a zero timeout.

can be used to expire flows prematurely in case of resource constraints.

Other reasons for expiring flow cache entries can be found in various flow exporter implementations:

- *Natural expiration* – A TCP packet with a `FIN` or `RST` flag set has been observed for a flow and therefore the flow is considered to have terminated.
- *Emergency expiration* – A number of flow entries are immediately expired when the flow cache becomes full.
- *Cache flush* – All flow cache entries have to be expired in unexpected situations, such as a significant change in flow exporter system time after time synchronization.

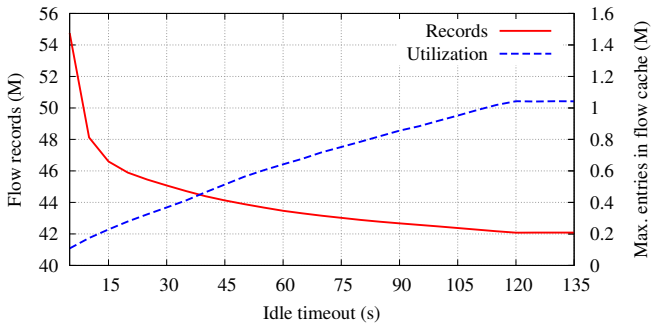
We survey how flow exporters handle flow cache entry expiration in practice in Section V-G.

The configured active and idle timeout values have impact on the total number of flow records exported for a particular dataset and the number of flow entries in the flow cache. On the one hand, using longer timeout values results in a higher aggregation of packets into flow records, which is generally positive and desirable to reduce the load on flow collectors. On the other hand, using longer timeout values means that it takes longer before a flow becomes visible to the Data Analysis stage.

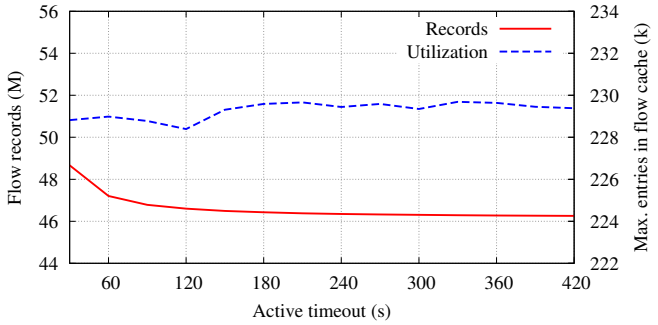
To illustrate the expiration behavior of a typical flow exporter, we have performed several experiments using the dataset presented in Section I-B on the impact of active and idle timeout values on 1) the number of resulting flow records, and 2) the maximum flow cache utilization. *nProbe*, an open-source flow exporter that will be discussed in Section V-G, has been used for exporting the flows without sampling. All experiments have been performed twice: Once by varying the active timeout value while maintaining a fixed idle timeout value, and once by varying the idle timeout value while maintaining a fixed active timeout value.

The experiment results are shown in Fig. 8. The figure shows the maximum number of concurrently used flow cache entries for various timeout values. Several conclusions can be derived from the experiment results. First, as shown in Fig. 8(a), an increasing idle timeout value results in fewer flow records, which is the case because of more packets being aggregated into the same flow record. This implies that flow entries stay in the flow cache for a longer time, resulting in a higher flow cache utilization. Second, the number of exported flow records and the maximum flow cache utilization stabilize for an increasing active timeout value, as shown in Fig. 8(b). This can be explained by the fact that most flow entries are expired by the idle timeout because of the very large active timeout value. Third, as soon as the idle timeout value equals the active timeout value (i.e., 120 seconds for our experiments), as shown in Fig. 8(a), the number of flow records and the flow cache utilization stabilize again, which is due to the fact that flow records are expired by the active timeout. We have also measured the impact of using natural expiration based on TCP flags and conclude that it barely affects the total number of flow records and the flow cache utilization.

Besides showing the relation between active and idle timeout behavior, the results in Fig. 8 provide insight into the



(a) Varying idle timeout values, active timeout = 120 seconds



(b) Varying active timeout values, idle timeout = 15 seconds

Fig. 8. Impact of timeouts on the aggregation of packets into flows and flow cache utilization.

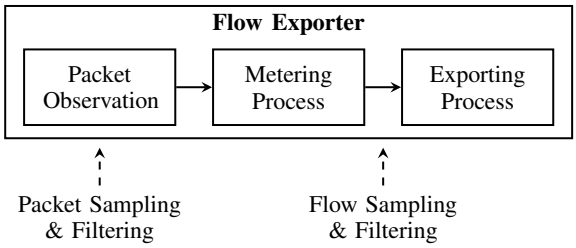


Fig. 9. Sampling & Filtering in a flow exporter.

minimum flow cache size required for monitoring the link in this specific example, which has a top throughput of roughly 2 Gbps. For example, given an active and idle timeout values of 120 and 15 seconds, respectively, the maximum flow cache utilization never exceeds 230k cache entries. More insights into flow cache overload and dimensioning are provided in Section VIII-A.

D. Flow Record Sampling & Filtering

Flow record sampling and filtering provide a means to select a subset of flow records, with the goal to reduce the processing requirements of the *Exporting Process* and all subsequent stages. In contrast to packet sampling and filtering, which are performed as part of the Packet Observation stage, flow record sampling and filtering functions are performed after the Metering Process and therefore work on flow records instead of packets. This is shown in Fig. 9. As a consequence, either all packets of a flow are accounted, or none.

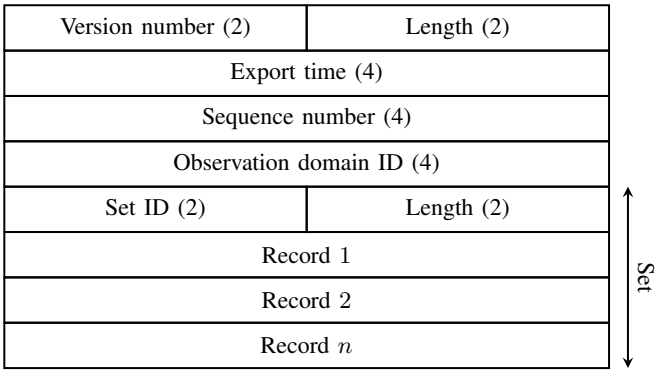


Fig. 10. IPFIX message (simplified) [1].

The techniques for performing flow record sampling and filtering are similar to packet sampling and filtering, which have been described in Section IV-D. We distinguish again between systematic sampling and random sampling [54]. Systematic sampling decides deterministically whether a flow record is selected (for example, every Nth flow record in periodic sampling). In contrast, with random sampling, flow records are selected in accordance to a random process. As for packet sampling, random sampling should be generally preferred over systematic sampling when in doubt about the characteristics of the traffic, because the latter can introduce unwanted correlations in the observed data.

Flow record filtering can be distinguished between *Property Match Filtering* and *Hash-Based Filtering* [54]. Property Match Filtering for flow records works similarly to Property Match Filtering for packets, but rather than filtering on packet attributes, filtering is performed on flow record attributes. It is particularly useful when only flow records for specific hosts, subnets, ports, etc. are of interest. With Hash-Based Filtering, flow records are selected if the hash value of their flow key lies within a predefined range of values. Hash-Based Filtering can be used for selecting a group of flow records from different observation points. Flow records from different observation points can be correlated because the flow key shared by packets belonging to the same flow results in the same hash value.

E. IPFIX Messages

A simplified version of the IPFIX message format [1] is shown in Fig. 10. The field size in bytes is shown for fields with a fixed size; other fields have a variable length. The first 16 bytes of the message form the message header and include a protocol version number, message length, export time and an *observation domain ID*. After the header come one or more *Sets*, which have an ID and a variable length, and can be of any of the following types:

- *Template Sets* contain one or more templates, used to describe the layout of Data Records.
- *Data Sets* are used for carrying exported Data Records (i.e., flow records).
- *Options Template Sets* are used for sending meta-data to flow collectors, such as control plane data or data applicable to multiple Data Records [55]. For example, Options Template Sets can be used to inform flow collectors about the flow keys used by the Metering Process.

Sets are composed of one or more records. The number of records in an IPFIX message is usually limited to avoid IP fragmentation. It is up to the Exporting Process to decide how many Records make up a message, while ensuring that the message size never exceeds the Maximum Transmission Unit (MTU) of a link (e.g., 1500 bytes) [1]. An exception to this rule is a situation in which IEs with variable lengths that exceed the link MTU are exported.

An example of a template, a corresponding Data Record, and a flow record is shown in Fig. 11. The template is shown at the top of the figure, and consists of an ID (257) and 9 IEs. A corresponding Data Record points at the appropriate template by listing its ID. This is mandatory to provide a means for flow collectors to associate Data Records with their templates. Also multiple flow records are included in the Data Record, which must adhere to the full set of IEs listed in the template.

TABLE II
COMPARISON OF TRANSPORT PROTOCOLS FOR IPFIX

	SCTP	TCP	UDP
Congestion awareness	+	+	–
Deployability	–	+	+
Graceful degradation	+	–	–
Reliability	+	+	–
Secure transport	+	+	–

F. Transport Protocols

After constructing an IPFIX message for transmission to a flow collector, a transport protocol has to be selected. A key feature of IPFIX is support for multiple transport protocols [1]. A comparison of transport protocols for IPFIX is provided in Table II, where '+' stands for *supported* or *good*, and '-' for *unsupported* or *poor*.

The Stream Control Transmission Protocol (SCTP) [56] is the mandatory transport protocol to implement for IPFIX. It provides a congestion-aware and sequential packet delivery service; packets are kept in sequence as with TCP, and packet boundaries are preserved as with UDP, i.e., the receiver can distinguish between individual packets, rather than a stream of bytes as with TCP. SCTP also provides multiple streams per connection, which can be used to avoid head-of-line blocking when multiple logical separate streams (e.g., one per template) are exported simultaneously [57]. The partial reliability extension [58] to SCTP provides further flexibility: The Exporting Process can cancel retransmission of unreceived datagrams after a given timeout. This allows graceful degradation via selective dropping of exported datagrams under high load, rather than overloading buffers with pending retransmissions.

Despite these advantages, SCTP is currently the least-deployed of the three supported protocols. The reason is primarily a practical one: IPFIX over SCTP can be difficult

[6]This IE has been abbreviated for the sake of space. The full IE name is shown in the template.

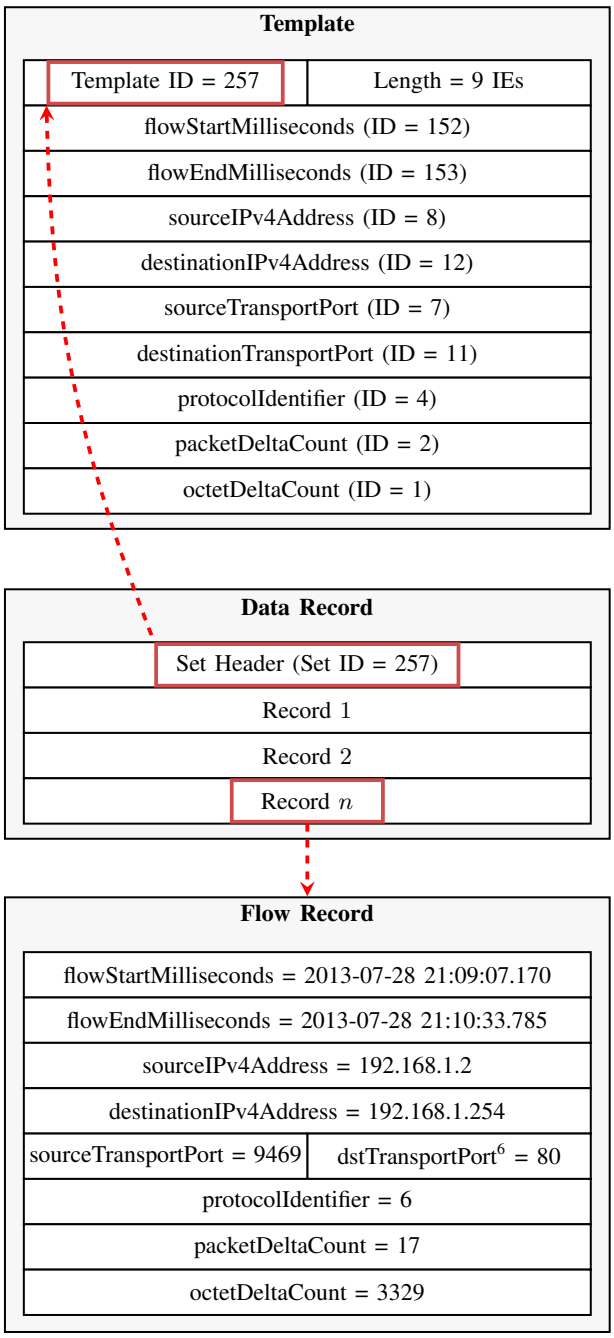


Fig. 11. Correlation between IPFIX data types (simplified) [1].

to implement, mainly because support for SCTP lags on other operating systems than Linux and BSD. Bindings to DTLS[7] for secure transport may also be hard to find in all but the most recent versions of TLS libraries. There are also deployment considerations. Since much more effort has gone into TCP stack optimization than SCTP stack optimization, the latter can be slower than TCP. It can also be difficult to send SCTP packets across the open Internet, as some middleboxes drop SCTP packets as having an unrecognized IP protocol number. Also, many Network Address Translation (NAT) devices often fail to support SCTP. However, given its advantages, we advocate using SCTP for flow export in every situation in which it is possible to do so.

IPFIX supports transport over TCP as well. TCP provides congestion-aware, reliable stream transport. It is widely implemented and, as such, it is very easy to implement IPFIX over TCP on most platforms. Bindings to TLS for secure transport are also widely available, which makes IPFIX over TLS over TCP the preferred transport for exporting flow records over the open Internet. The primary problem with IPFIX over TCP is that TCP does not degrade gracefully in overload situations. Specifically, the TCP receiver window mechanism limits the Exporting Process' sending rate when the Collecting Process is not ready to receive, thereby locking the rate of export to the rate of collection. This pushes buffering back to the Exporting Process, which is generally the least able to buffer datagrams. Careful selection of TCP socket receive buffer sizes and careful implementation of the Collecting Process can mitigate this problem, but those implementing Collecting Processes should be aware of it.

The most widely implemented and deployed transport protocol for flow export is UDP. UDP has the advantage of being easy to implement even in hardware Exporting Processes. It incurs almost no overhead, but on its turn provides almost no service: "Best-effort" (or "unreliable") delivery of packets without congestion awareness. As a consequence, UDP should be used for flow export with care. The lack of any congestion awareness means that high-volume export may incur significant loss. The lack of flow control means that Collecting Processes must use very large socket buffers to handle bursts of flow records. As the volume of exported flow records increases dramatically during Denial-of-Service (DoS) attacks or other incidents involving large numbers of very short flows, the lack of flow control also may make UDP futile for measuring such incidents. Another serious problem concerns templates. On UDP, Exporting Processes must periodically resend templates to ensure that Collecting Processes have received them. While IPFIX does provide a sequence numbering facility to allow a Collecting Process to roughly estimate how many flow records have been lost during export over UDP, this does not protect templates. A Collecting Process that loses a template, or that restarts in the middle of an export, may be unable to interpret any flow records until the next template retransmission.

A fourth method provided by IPFIX is the IPFIX File Format [59]. IPFIX messages are grouped together into files, which can be stored and transported using any of the various protocols that deal with files (e.g., SSH, HTTP, FTP and NFS). File transport is not particularly interoperable and therefore not recommended in general. However, it may be worth considering in specific cases, such as making IPFIX flow data widely available via a well-known URL.

G. Open-Source Tools & Commercial Appliances

Open-source and commercial flow exporters existing to date can be classified into two types: hardware-based and software-based. Hardware-based exporters can usually achieve higher throughputs, while software-based ones provide greater flexibility in terms of functionality. Software-based exporters are

[7]DTLS is an implementation of TLS for transmission over datagram transport protocols, such as UDP and SCTP.

TABLE III
OPEN-SOURCE FLOW EXPORTERS

		ipt-netflow	softflowd	nProbe [60]	pmacct [61]	QoF	Vermont [62]	YAF [63]
Version		1.8	0.9.9	6.13	0.14.3	0.9.0[1]	1.0.0a	2.4.0
Application awareness				✓				✓
Flow cache entry expiration		Active timeout, idle timeout						
		TCP `FIN/RST`						TCP `FIN/RST`
Flow key		Source IP address, destination IP address, source port number, destination port number						
		IP protocol number, IP ToS, SNMP interface ID	IP protocol number	IP protocol number, IP ToS, VLAN ID	IP protocol number			VLAN ID, IP protocol number
Flow sampling				✓	✓			
Packet sampling			✓	✓	✓		✓	
NetFlow v5		✓	✓	✓	✓		✓	
NetFlow v9			✓	✓	✓		✓	
IPFIX	Bidirectional flows			✓	✓	✓	✓	✓
	Structured data (RFC 6313)							✓
	Enterprise-specific IEs			Application information, performance metrics, geolocation information, TCP metrics, plugins		Performance metrics		Application information
	Options templates			✓	✓	✓	✓	✓
	Transport protocols			SCTP, TCP, UDP	UDP	SCTP, TCP, UDP, file	SCTP, TCP, UDP, file	
	Variable-length encoding			✓	✓		✓	✓

[1] Only a pre-release version was available at the time of writing.

also less costly. Another classification divides flow exporters into open-source and commercial exporters. In this section, we provide both an overview of some available open-source and commercial flow exporters, and a hands-on guide for those selecting a new flow exporter.

When selecting a flow exporter for deployment, it is important to verify the following criteria:

- *Throughput* – Throughput is one of the most important properties of a flow exporter, as it shows how many flows, packets and bits can be processed per second. Most vendors and developers express throughput in *Gbps*, without specifying the number of packets that can be handled per second, which leads to some ambiguity. For example, *10 Gbps* can mean 14.88 Mpkt/s, calculated based on the minimum allowed frame size for Ethernet, or 812.84 kpkt/s, calculated based on the maximum allowed frame size. Moreover, the provided performance indications often refer to the case where most packets can be mapped to existing flow cache entries. The rate at which the device can create new entries in the flow cache is usually significantly lower. In addition, many vendors and developers express the throughput of their flow exporters for the case in which only a limited set of IEs is used within the Metering Process. It is often the case that the advertized throughput cannot be achieved anymore when additional IEs are enabled.
- *Flow cache size* – As many flow exporters come with a fixed-size flow cache, it is important to have an understanding of how many flows transit in the network, to avoid flow cache under-dimensioning. The availability of expiration policies should also be checked, as they affect the flow cache utilization.
- *Supported IEs and accuracy thereof* – Many flow exporters, mostly older and hardware-based, support only a limited set of IEs; MAC addresses, VLAN tags, MPLS labels, TCP flags, or application information often cannot be exported. In addition, it should be verified whether the claimed accuracy of IEs is actually provided. For example, it is shown in [64] that many high-end packet forwarding devices do not convey TCP flag information in flow data, although the required IE (ID 6) is actually supported. Also the precision of exported timestamps varies from exporter to exporter; IPFIX supports timestamps ranging from second to nanosecond precision, and care should be taken that the timestamp precision matches the accuracy prescribed by the IE. Also, the

timestamp precision should meet the requirements of the Data Analysis stage. More information on the accuracy of flow data is provided in Section VIII-D.

In addition to these criteria, one may consider another criterion that is rapidly gaining importance: application awareness. Application awareness in flow export is relatively young, but given that it increases visibility in network traffic and the fact that the number of analysis applications supporting it is increasing as well, it should be considered when selecting a flow exporter.

We have compiled a list of open-source flow exporters in Table III. All presented flow exporters are software-based and have been updated at least once since 2008. The table compares both general flow exporter properties (upper part) and properties specific to the various flow export protocols (lower part), and can be summarized as follows:

- The flow keys used by the various flow exporters do all include IP addresses and port numbers, but differ greatly with respect to the remaining fields. A surprising observation that can be made is that the typical *5-tuple* and *7-tuple*[8], which are commonly-used terms for describing IP flows, are only used by four out of seven exporters. In addition, none of the tools supporting IPFIX allows for the flexible definition of flow keys (not shown in Table III).
- All tools support NetFlow v5, NetFlow v9, or both, except for *YAF* and *QoF*, which have been designed specifically for IPFIX-compliance. *ipt-netflow* and *softflowd* do not support IPFIX at all.
- Although "IPFIX support" is advertized for most flow exporters, some IPFIX-specific features are still unsupported. Support for bidirectional flows, options templates and variable-length encoding is widely available, while only YAF supports structured data. This may be due to the fact that structured data has been standardized only in 2011. Finally, SCTP support is provided by all IPFIX flow exporters but *pmacct*, although operating system support is often still lacking.
- Packet sampling is supported by most tools, while flow sampling is only available in *nProbe* and *pmacct*.
- *nProbe*, *QoF* and *YAF* export several enterprise-specific IEs, mostly targeted at application identification and latency measurements.

Besides the open-source flow exporters listed in Table III, there are flow exporters available that do not export flow data using NetFlow or IPFIX. These exporters write the flow data directly to text or binary files, for example, without the involvement of a Data Collection stage. A well-known example is *tstat* [65], which has a strong focus on application awareness and performance metrics. Since this paper is a tutorial on NetFlow and IPFIX, we consider such tools out of the scope of this paper.

The market of commercial flow exporters consists mostly of appliance products: packet forwarding devices (e.g., routers and switches), firewalls, and dedicated flow probes. Forwarding devices and firewalls are often already available in networks, and if they have flow export support, the step to enable this functionality is relatively small. These devices usually export the vast majority of flows in hardware using Application-Specific Integrated Circuits (ASICs), so that flow export does not consume costly CPU time. Although this results in a very high performance, it also has the disadvantages of being more expensive and usually less flexible than software-based solutions. For example, it is almost impossible to fix bugs and introduce new features in such tailored hardware.

Commercial flow probes are typically part of a flow monitoring system including collection and analysis tools from the same vendor and overcome several limitations of packet forwarding devices with flow export capabilities. They usually come in two types: fully software-based and hardware-accelerated. Software-based probes (mostly Linux-based) are often sold on commodity hardware and are therefore much cheaper than hardware-based flow exporters. They can be equipped with hardware-acceleration to achieve line-rate processing. Hardware-acceleration is usually performed by special cards with FPGAs or commodity NICs with special firmware. Even hardware-accelerated solutions rely on software-based flow exporters, as the acceleration is purely used for packet timestamping, filtering and packet distribution over CPU cores. Given their architecture, flow probes are more flexible when it comes to bug fixing and introducing new features, compared to packet forwarding devices.

Many vendors have their own implementation of NetFlow, while others follow with their own NetFlow-alike technologies, such as Juniper's J-Flow. Since its standardization by the IETF, IPFIX is becoming dominant for many vendors. Experience has shown, however, that commercial solutions so far do not provide full IPFIX compliance, although many of them claim to have "IPFIX support". A list of recognized commercial flow exporters is provided in Table IV, where we evaluate these based on export protocol support, application awareness, and the main selling features (i.e., what the main features advertized by the vendor are). Since we have no access to all listed commercial flow exporters, we compare these devices only by means of information made public by the vendor by means of feature specifications, manuals, etc. We have included all available flow probes, those packet forwarding devices that (partly) support hardware-based flow export, and those firewalls exporting flows. It is clear that vendors of flow probes try to be as flexible as possible with respect to integration into existing environments (e.g., flow collectors), as all surveyed appliances support NetFlow v5, NetFlow v9 and IPFIX. Also application awareness is widely supported among flow probes nowadays. Less flexibility is shown by the listed packet forwarding devices, mostly because of the fact that they perform flow export (partly) in hardware. Finally, the firewall appliances show a clear focus on security-oriented information export by means of application awareness.

[8]The *5-tuple* consists of IP addresses, port numbers and IP protocol number. The *7-tuple* additionally includes IP ToS and SNMP input interface ID.

TABLE IV
COMMERCIAL FLOW EXPORTERS

	Vendor	Product	NetFlow v5	NetFlow v9	IPFIX	Application awareness	Main selling point(s)
Probes	Cisco	NetFlow Generation Appliance	✓	✓	✓	✓	Cross-device flow export in high-speed networks
	Emulex (Endace)	Endace NetFlow Generator	✓	✓	✓		High-performance flow export based on Endace DAG Cards
	INVEA-TECH	FlowMon Probe	✓	✓	✓	✓	High-performance flow export (both software-based and hardware-accelerated)
	Lancope	StealthWatch FlowSensor	✓	✓	✓	✓	Flow export with focus on application awareness and performance metrics
	ntop	nBox NetFlow/IPFIX	✓	✓	✓	✓	Commercial versions of open-source tools (nProbe, ntop)
Forwarding devices	Cisco	Cisco IOS Flexible NetFlow	✓	✓	✓	✓	Application awareness
	Enterasys	N-Series, S-Series	✓	✓			Flexible flow export based on ASIC
	Extreme Networks	ExtremeXOS			✓		Flow export on L2 through L4
	Juniper	Junos J-Flow	✓	✓	✓		Flow export on L2 through L4
Firewalls	Barracuda Networks	Barracuda NG Firewall			✓	✓	Audit logs and HTTP proxy reporting using IPFIX
	Dell	Sonic Wall Next-Generation Firewall	✓	✓	✓	✓	Application awareness
	Palo Alto Networks	Next-Generation Firewall		✓		✓	Flow export with specific IEs, such as application and user IDs

VI. DATA COLLECTION

Flow collectors are an integral part of flow monitoring setups, as they receive, store, and pre-process[9] flow data from one or more flow exporters in the network. Data collection is performed by one or more Collecting Processes within flow collectors. Common pre-processing tasks are data compression [59], [66], aggregation [67], data anonymization, filtering, and summary generation.

In this section, we discuss the most important characteristics of flow collectors. We start by describing the various formats in which flow data can be stored in Section VI-A. After that, in Section VI-B, we provide best-practices in the field of data anonymization. Anonymization is a type of data obfuscation that ensures anonymity of individuals and prevents tracking individual activity. We close this section with Section VI-C, where we provide an extensive analysis of open-source and commercial flow collection implementations.

A. Storage Formats

The functionality and performance provided by flow collectors depend strongly on the underlying data storage format, as this defines how and at which speed data can be read and written. This section discusses and compares the various available storage formats, which should allow one to choose a flow collector that satisfies the requirements of a particular setup or application area. We can distinguish two types of storage formats:

- *Volatile* – Volatile storage is performed in-memory and therefore very fast. It can be useful for data processing or caching, before it is written to persistent storage.
- *Persistent* – Persistent storage is used for storing data for a longer time and usually has a larger capacity. However, it is significantly slower than volatile storage.

Although flow data often has to be stored for a long time (e.g., to comply with data retention laws), it can be useful to keep data in-memory. This is mostly the case when flow data has to be analyzed on-the-fly, and only results have to be stored. In those situations, one can benefit from the high performance of volatile storage. An example use case is the generation of a time-series in which only the time-series data itself has to be stored.

When data has to be stored beyond the time needed for processing, it has to be moved to persistent storage. This, however, results in a bottleneck, due to the difference in speed between volatile and persistent storage. Depending on the system facilitating the flow collection, one may consider to compress data before moving it to persistent storage (more details are provided in Section VIII-C). We distinguish between the following types of persistent storage:

- *Flat files* – Flat file storage is usually very fast when reading and writing files, while providing limited query facilities [68]. Examples of flat file storage are binary and text files.
- *Row-oriented databases* – Row-oriented databases store data in tables by means of rows and are frequently used in Database Management Systems (DBMSs), such as MySQL[10], PostgreSQL[11], and Microsoft SQL Server[12]. Accessing the data is therefore done by reading the full rows, even though only part of the data may be needed to answer a query.

[9]We talk about *pre-processing* here, as we assume that *processing* is done in the Data Analysis stage.

[10]https://www.mysql.com/
[11]http://www.postgresql.org/
[12]http://www.microsoft.com/sql/

TABLE V
COMPARISON OF DATA STORAGE FORMATS

	Flat files	Row-oriented databases	Column-oriented databases
Disk space	+	–	0
Insertion performance	+	–	0
Portability	– (binary), + (text)	–	–
Query flexibility	–	+	+
Query performance	+ (binary), – (text)	–	+

- *Column-oriented databases* – Column-oriented databases, such as FastBit[13], store data by column rather than by row. Only fields that are necessary for answering a query are therefore accessed.

A comparison of these data storage formats is shown in Table V, where '+' stands for *good*, '–' for *poor*, and '0' for *average*. We evaluate each format based on disk space requirements, insertion performance, portability, query flexibility and query performance. We consider *nfdump* a representative option for binary flat file storage, MySQL for row-oriented databases, and FastBit for column-oriented databases.

In terms of disk space, flat files have a clear advantage above the database-based approaches. This is mainly due to the fact that row- and column-oriented databases usually need indexes for shorter query response times, which consume more disk space on top of the "raw" dataset. For example, it is described in [69] that MySQL with indexes needs almost twice the capacity of *nfdump* for a particular dataset. For the case of FastBit, it is shown to be less capacity-intensive than MySQL (depending on its configuration) [68], but more than *nfdump*. High compression rates can be achieved since data in a particular column is usually very similar, i.e., homogeneous. The highest insertion performance can be achieved using flat files, as new data can be simply added to the end of a file, without any additional management overhead, such as updating indexes in the case of MySQL. When it comes to portability, text-based flat files have the clear advantage of being readable by many tools on any system. However, flat files usually provide only limited query language vocabulary, which makes databases more flexible in terms of possible queries.

In contrast to database-based approaches, flat file storage is usually not indexed; sequential scans over datasets are therefore unavoidable. However, since many flat file storages create smaller files at regular intervals, this can be considered a coarse time-based index that limits the size of sequential scans by selecting fewer input files in a query. Several works have compared the performance of the various data storage formats in the context of flow data collection. The performance of binary flat files and MySQL is compared in [69], where query response times are measured for a set of queries on subsets of a single dataset. The authors show that binary storage outperforms MySQL-based storage in all tested scenarios and advocate the use of binary storage when short query response times are required. A similar methodology has been used in [70], where the performance of FastBit is compared with binary flat files. It is shown that FastBit easily outperforms binary storage in terms of query response times, which is explained by the fact that FastBit only reads columns that are needed for a query. This results in fewer I/O operations. The performance of FastBit- and MySQL has been compared in [68], where the authors conclude that FastBit-based storage is at least an order of magnitude faster than MySQL.

The performance of the described approaches can generally be improved by distributing flow data over multiple devices. For example, it is a common practice to use storage based on a Redundant Array of Independent Disks (RAID) in a flow collector, which ensures data distribution over multiple hard drives in a transparent fashion. Even more performance improvements can be achieved by deploying multiple flow collectors and distributing the data between them. This, however, requires some management system that decides how data is distributed (for an example, see [71]).

B. Data Anonymization

Flow data traditionally has a significant privacy protection advantage over raw or sampled packet traces: Since flow data generally does not contain any payload, the content of end-user communications is protected. However, flows can still be used to identify individuals and track individual activity and, as such, the collection and analysis of flow data can pose severe risks for the privacy of end users. The legal and regulatory aspects of this privacy risk, and requirements to mitigate it are out of scope for this work – these are largely a matter of national law, and can vary widely from jurisdiction to jurisdiction. Instead of surveying the landscape of data protection laws, we make a general simplifying assumption that IP addresses can be used to identify individuals and as such should be protected. Other information available in flows can be used to disambiguate flows and therefore may be used to profile end users or to break IP address anonymization.

Best practices for trace data anonymization are laid out by CAIDA in [75], drawing on the state of the art in anonymization techniques surveyed in [76]. The key tradeoff in IP address anonymization is between privacy risk and data utility. There is no 'one-size-fits-all' flow data anonymization strategy, as data utility is also dependent on the type of analysis being done. For example, simply removing IP address information carries with it the lowest risk of identification, but also makes the data useless for anything requiring linkage of flows to hosts; for simple statistics on flow durations and volumes, for example, such data can however still be useful.

In the more general case, since networks are structured, a structure-preserving anonymization technique such as the Crypto-PAn algorithm [77] allows anonymized IP addresses to be grouped by prefix into anonymized networks. This preserves significant utility for analysis, at the cost of restricting the space of possible anonymized addresses for a given real address, making profiling attacks against address anonymization easier. Even given this tradeoff, the significantly increased

[13]https://sdm.lbl.gov/fastbit/

TABLE VI
OPEN-SOURCE FLOW COLLECTORS

		Argus	flowd	IPFIXcol [72]	nfdump [73]	nProbe [60]	pmacct [61]	SiLK [74]	Vermont [62]
	Version	3.0.7.5	0.9.1	0.6.0	1.6.10	6.13	0.14.3	3.7.1	1.0.0.a
	Anonymization	✓			✓			✓	✓
	Storage formats	Flat files (binary, text), row-oriented DB	Flat files (binary)	Column-oriented DB	Flat files (binary)	Flat files (binary, text), row-oriented DB	Flat files (text), row-oriented DB	Flat files (binary)	Row-oriented DB
	NetFlow v5	✓	✓	✓	✓	✓	✓	✓	
	NetFlow v9	✓	✓	✓	✓	✓	✓	✓	✓
IPFIX	Bidirectional flows			✓		✓	✓		✓
	Enterprise-specific IEs			Any		Application information, performance metrics, geolocation information, plugins		Application information	
	Options templates			✓	✓	✓	✓	✓	✓
	Structured data (RFC 6313)							✓	
	Transport protocols			SCTP, TCP, UDP, file	UDP	SCTP, TCP, UDP	UDP	SCTP, TCP, UDP	SCTP, TCP, UDP, file
	Variable length encoding			✓		✓	✓		✓

utility of the results leads to a recommendation for Crypto-PAn.

Given the restricted space of solutions to the anonymization problem, it has become apparent that unrestricted publication of anonymized datasets is probably not a tenable approach to the sharing of flow data [78], as attacks against anonymization techniques scale more easily than strengthening these techniques while maintaining utility. Technical approaches to data protection are therefore only one part of the puzzle; non-technical protection, such as sharing of data in vetted communities of researchers, or analysis architectures whereby analysis code is sent to a data repository and only results are returned, must also play a part in preserving the privacy of network end-users.

C. Open-Source Tools & Commercial Appliances

When selecting a flow collector for deployment, regardless of whether an open-source or commercial one is selected, it is important to verify the following criteria:

- *Performance* – The performance of flow collectors is usually expressed in terms of the number of flow records that can be received, pre-processed and stored per second.
- *Storage format* – The storage format used by the flow collector determines how the stored flow data can be accessed.
- *Export protocol features* – It is essential for a flow collector to support the same export protocol features as the used flow exporter, such as data encodings, transport protocols, and IEs. A flow collector that does not support full flow stream collection (i.e., all exported elements in the received data stream can be processed and stored) may lead to data loss. Special attention should be paid when non-traditional IEs are used, such as IEs related to application awareness or enterprise-specific IEs. Guidelines on which flow exporter features to use in which situation are provided in Section V.
- *Processing delay* – Data analysis always has to wait for a flow collector to finish processing the flow data. The shorter the processing delays are, the more timely data analysis can take place within the Data Analysis stage.
- *Flow record deduplication* – This technique eliminates flow record duplicates in a dataset, which can be the result of a flow being exported at multiple observation points in a network. Flow record duplicates lead to erroneous accounting and suboptimal security analysis, for example.
- *Integration with other systems* – Since flow data is often used as a complement to other network monitoring and management technologies, it is important that enterprise flow collectors provide multiple integration interfaces for technologies as SNMP, syslog, REST API, etc.

We have compiled a list of open-source flow collectors that have been updated at least once since 2008 in Table VI. Several observations can be made. First, all available storage formats discussed before are supported by at least one collector. This can make the selection of a flow collector easier, in situations where a particular storage format is a hard requirement. Second, IPFIX is claimed to be widely supported, although some IPFIX-specific features, such as structured data export, are barely available. Third, only three flow collectors support flow data anonymization.

TABLE VII
COMMERCIAL FLOW COLLECTORS

	Arbor Networks	Fluke Networks	INVEA–TECH	Lancope	Manage-Engine	Plixer	Riverbed	SevOne	Solar–Winds
Flow records per second	250k	40k	200k	120k	10k	100k	23k	250k	40k
Storage format	(proprietary)	Row-oriented database	Flat files (binary)	Column-oriented database	Row-oriented database				
NetFlow v5	✓	✓	✓	✓	✓	✓	✓	✓	✓
NetFlow v9	✓	✓	✓	✓	✓	✓	✓	✓	✓
IPFIX	✓	✓	✓	✓	✓	✓	✓	✓	✓

Most open-source flow collectors do not come with any specification of their performance in terms of the number of flow records that can be processed per second. In cases where it is specified, however, it is rather a performance indication of what the developer has achieved on test systems, rather than a guaranteed performance. This is mainly because the performance of flow collectors strongly depends on the selected storage format and the performance of the deployment machine's storage subsystem. We therefore have not included any performance indication in Table VI. Also *flow record deduplication* and *integration with other systems* are less common in open-source flow collectors, and have therefore been left out of the comparison. Finally, it should be noted that most flow collectors in Table VI have no processing delay, or that it is configurable. For example, *nfdump* uses intervals of five minutes by default.

A list of major vendors on the market of commercial flow collector appliances is shown in Table VII. None of these vendors sells appliances that solely perform flow data collection; they all come with some sort of analysis and reporting functionality instead, to provide easy-to-use and integrated flow solutions. In this section, we focus on the collection-related aspects of these appliances. The analysis and reporting functionality, however, will be discussed in Section VII.

In contrast to open-source flow collectors, the performance in terms of flow records per second that can be processed is always specified for commercial appliances. This is because commercial flow collectors are predominantly sold as an appliance, a dedicated machine with a RAID-based setup for performance and data redundancy, and flow collection software pre-installed. Vendors are able to select the underlying hardware, measure the overall performance, and provide performance guarantees, independent of the deployment setup. All vendors listed in Table VII sell appliances that support high traffic volumes (i.e., more than 10k flow records per second) and data collection from multiple sources, and have disk space for long-term data retention. Most of them use row-based databases, such as MySQL, PostgreSQL, and Microsoft SQL Server, although Lancope, for example, uses a column-oriented database that is optimized for reading operations, useful for data analysis in a later processing stage. An interesting observation with respect to the storage formats used in flow collection appliances, is that the fastest appliances rely on row-based DBMSs, while it has been discussed in Section VI-A to be one of the slower formats in the context of flow collection. This can be explained by the fact that appliances mostly use optimized DBMS setups, while DBMSs in a scientific context are often out-of-the-box installations. In addition, appliances heavily rely on volatile storage for pre-processing and Solid-State Drives (SSDs) for permanent storage.

VII. DATA ANALYSIS

Data Analysis is the final stage in a flow monitoring setup, where the results of all previous stages come together. We distinguish between three application areas for data analysis, which are widely used for classifying analysis software: 1) Flow Analysis & Reporting, 2) Threat Detection, and 3) Performance Monitoring. These areas will be discussed in Section VII-A, VII-B, and VII-C, respectively. We do this by explaining practical use cases, alternatives, and recent advances for each area. After that, we provide an extensive analysis of open-source and commercial flow data analysis software in Section VII-D. The goal of this section is to provide a glimpse what can be done with flow data.

For further reading, one may consider the IPFIX applicability statement issued by the IETF in [79], and the survey on network flow applications provided in [80].

A. Flow Analysis & Reporting

Given that flow export devices are commonly deployed at strategical locations in a network where traffic from a large number of hosts can be observed, the resulting data provides a comprehensive set of connections summaries. Flow Analysis & Reporting is the most basic functionality provided by flow analysis applications and typically provides the following functionality:

- *Browsing and filtering flow data*.
- *Statistics overview* – The most common statistics are for *top-talkers*, i.e., those hosts, autonomous systems or services that exchanged most traffic.
- *Reporting and alerting* – A commonly used reporting application is bandwidth reporting, i.e., which user/customer exchanged how much traffic. Alerting can be used when traffic thresholds are exceeded (e.g., when hosts are generating a suspicious number of connections) or hosts are communicating using unwanted applications or protocols.

A typical example of an application providing this functionality is *NfSen*[14], a popular open-source framework, featuring a Web-interface, built around the flow collection toolkit *nfdump*. It can be used in many situations, such as finding spamming hosts, routing problems, and misconfigured services (e.g., DNS). In the remainder of this subsection, we discuss how *NfSen* can be used for both manual flow data inspection and automatic reporting. For further reading, we recommend the tutorial on NfSen provided in [81].

The usual start for analyzing flow data is by observing the traffic graphs provided as part of the *Dashboard*, which provides insight in the traffic behavior in terms of flows, packets and bytes. Peaks in those graphs signal that the traffic behaves different from what is considered 'normal'. Whether this is an indication of something malicious or purely benign, can now be investigated by retrieving flow data statistics for the timeframe in which the anomaly was active. This reveals which hosts have been *top-talkers*. When top-talkers have been identified based on the number of flows they have generated, for example, it is not uncommon to identify sources of network scans, brute-force attacks, or Distributed DoS (DDoS) attacks, as these kind of attacks often result in many small flows [82]. After identification, raw flow data can be retrieved and analyzed to learn the actual nature of the anomalous traffic.

To automate some analysis options, *NfSen* provides reporting functionality by means of alerts. Alerts can be configured based on thresholds for nearly any traffic characteristic that can be expressed in terms of the number of flows, packets and bytes, and filters. This helps network managers to be aware of problems in the network as early as possible. Taking the scenario of hosts generating an abnormal number of flows, one could consider configuring a threshold based on the number of flows generated per time interval. This can inform network administrators of (large) attacks or other misuses targeting the monitored network and hosts.

The functionality of *NfSen* can be extended by means of plugins[15]. These plugins can process raw flow data and visualize the results in a Web interface. As such, *NfSen* can be extended to include Threat Detection or Performance Monitoring functionality. An example of a plugin for *NfSen* that provides Flow Analysis & Reporting functionality is SURFmap [83], a network monitoring tool based on the Google Maps API. It adds a geographical dimension to flow data and shows the flow data on a map.

B. Threat Detection

When flow data is used for threat detection, we can distinguish between roughly two types of uses. First, flow data may be used purely for analyzing which host has communicated with which each other host (i.e., forensics), potentially including summaries of the number of packet and bytes involved, the number of connections, etc. The second utilizes the definition of a flow for analyzing certain types of threats, which allows for modeling threats in terms of network behavior. In the remainder of this section, we discuss an example of both types.

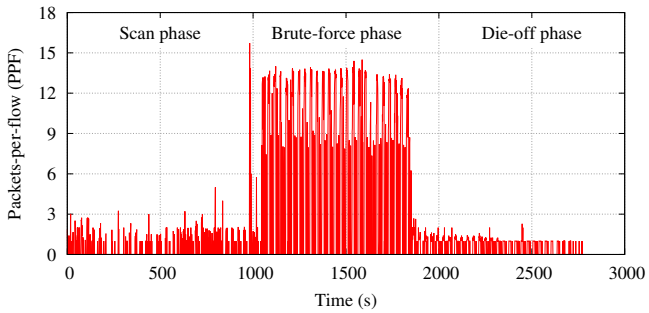


Fig. 12. Time-series of the number of PPF of an SSH dictionary attack (based on [86]).

The central observation points at which flow export devices are usually deployed make flow export especially useful for the detection of the following attacks and malwares [84]: DDoS attacks, network scans, worm spreading, and botnet communication. The commonality between these attacks is that they affect metrics that can be directly derived from flow records, such as the volume of traffic in terms of packets and bytes, the number of active flows in a certain time interval, suspicious port numbers commonly used by worms, and suspicious destination hosts for traffic.

The identification of suspicious destination hosts for traffic is usually done by means of *IP reputation lists*, or *blacklists*. IP reputation lists are lists of IP addresses that have been identified as sources of malicious activities. For example, they may have sent SPAM messages, hosted malware, or taken part in a botnet infrastructure. Flow data can be easily combined with IP reputation lists by checking whether the source or destination addresses of a record have been listed as offenders. This technique does however not only identify threats coming from outside the network perimeter; IP reputation lists can also help in the detection of Advanced Persistent Threats (APTs), for example. APTs[16] are modern attacks that combine a high degree of stealthiness, long term planning and a multiplicity of attack vectors. They typically target governmental and commercial entities, and aim at gaining a stronghold in the target network, for example, for cyber-espionage. By analyzing connections from the local network to external hosts with a poor reputation, APTs and other suspicious activities like botnets may be identified.

The second use of flow data for threat detection utilizes the common definition of a flow, to identify certain types of attacks. We discuss this by means of an example: Secure Shell (SSH). SSH provides secure remote access to a UNIX-based machine, and is a frequently-used target of *dictionary attacks*. These attacks use lists with often-used username and password combinations, which are tried on SSH daemons by means of brute-force. Once a remote machine has been compromised, the attacker gains control of it and can misuse it for all kinds of malicious purposes.

[14]http://nfsen.sourceforge.net/

[15]A list of plugins for *NfSen* is maintained at https://sourceforge.net/apps/trac/nfsen-plugins/.

[16]An APT that gained much media attention is APT1 [85].

```
Start      Source              Destination   Flags Pkts
-------------------------------------------------------
03:07:21  87.2.34.3:46682  36.128.7.9:22   ....S.   1
03:09:36  87.2.34.3:59459  36.128.7.9:22   .AP.SF  12
03:09:39  87.2.34.3:59973  36.128.7.9:22   .AP.SF  12
03:09:42  87.2.34.3:60318  36.128.7.9:22   .AP.SF  12
                          ...
```

Fig. 13. Dictionary attack pattern in flow records (simplified). The IP addresses have been anonymized.

Despite the fact that SSH traffic is encrypted, SSH dictionary attacks can be easily detected using flow analysis because of a typical attack pattern: Many credentials are tested subsequently and SSH daemons close the connections after a fixed number of login attempts, resulting in many TCP connections with similar size in terms of packets. An example of this is shown in Fig. 12, where three attack phases can be identified: 1) *scan phase*, where an attacker probes for active SSH daemons ($t < 1000$), 2) *brute-force phase*, where the actual dictionary attack is performed ($1000 \leq t \leq 1900$), and 3) *die-off phase*, where residual traffic may be exchanged between attacker and target after a compromise ($t > 1900$), for example. Important here is the observation that the *scan phase* shows a low number of Packets-Per-Flow (PPF), while the *brute-force phase* shows a significantly higher number of PPF.

The pattern with respect to PPF described before can also be identified in the corresponding flow records, as shown in Fig. 13; the first flow record (source port 46682) clearly indicates a scan, while the other flow records match the pattern of login attempts. The detection of these patterns can be performed in an automated fashion [87], as is done by the Intrusion Detection System (IDS) SSHCure[17], for example. SSHCure is completely flow-based, and is able to identify various flavors of the flow pattern discussed before, e.g., with a different number of PPF. This demonstrates two main advantages of flow-based intrusion detection: It works in encrypted environments, as it does not rely on packet payloads, and its light-weight nature allows for analyzing even backbone links with thousands of flows per second. For further reading, we recommend the survey on flow-based intrusion detection provided in [84].

C. Performance Monitoring

Performance monitoring aims at observing the status of services running on the network. Typical metrics that such data analysis applications report include Round-Trip-Time (RTT), delay, jitter, response time, packet loss and bandwidth usage. Performance monitoring applications post-process flow data and show a set of metrics per target service, to verify Service-Level Agreement (SLA) compliance and, ultimately, reveal network events and their impact on end-user experience.

As for the other types of data analysis, the greatest strength of monitoring performance using flow measurements comes from the strategical vantage points from where flow measurements are usually taken. As a comparison, monitoring

[17]https://sshcure.sourceforge.net/

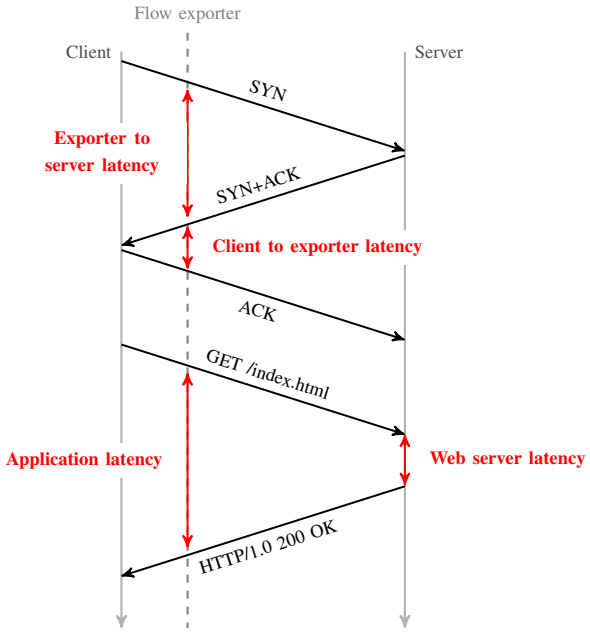


Fig. 14. Estimating Web server latency (based on [88]).

applications by means of *client instrumentation* requires the installation of agents in client devices, which makes the measurement environment not only less convenient to set up, but also harder to be managed. However, as we will exemplify next, flow measurements sometimes provide only a coarse approximation of common performance metrics, since such metrics are generally not directly measured and exported.

Flow-based performance monitoring applications can be roughly divided into two groups. A first group of applications tries to estimate performance metrics, such as service availability [89], RTT or one-way delay [90], by post-processing the IEs that are commonly exported by flow exporters (see Section V-A). The main advantage of such an approach is that no customization is needed in either flow exporters or flow collectors, i.e., to export and collect enterprise-specific IEs. However, as high-level performance metrics are not part of the most common list of IEs, the precision of the reported metrics might not be sufficient in certain circumstances. As an example, the estimation of one-way delays using NetFlow records has been evaluated in [90]. The one-way delay between two routers is estimated from the difference between the flow start times reported by the routers for the same flow. When applying this approach to empirical data, it quickly turns out that the results are significantly affected by timing errors. This is partly compensated for by introducing a calibration phase, in which offline flow measurements are used to create exporter-specific profiles, containing information on clock offsets and skews, timer resolution, among others (see also Section VIII-D).

A completely different approach is taken by a second group of analysis applications, which rely on extensions for flow exporters that extract performance metrics. We illustrate that by showing how *nProbe* can be used for exporting the latency of Web servers. *nProbe* and other flow exporters that provide

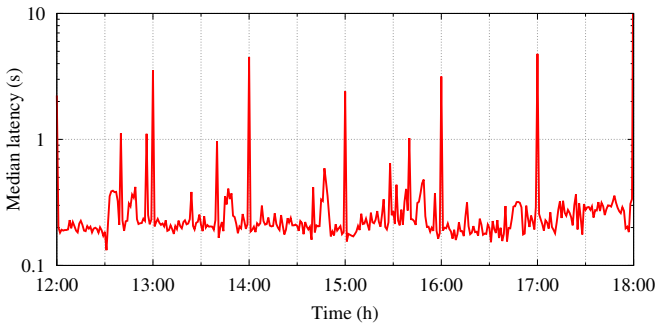


Fig. 15. Monitoring a Web server using flow measurements.

advanced application awareness (see Section V-G) implement similar functionality to monitor other protocols as well, such as DNS. By analyzing also packet payloads, *nProbe* is able to identify HTTP traffic and export enterprise-specific IEs that report the network latency from client to the flow exporter, the network latency from the flow exporter to servers and the total application latencies. The semantic of these metrics is depicted in Fig. 14. The Web server latency can be obtained by subtracting the server network latency from the total application latency.

Fig. 15 illustrates how these IEs are useful for monitoring the performance of Web servers. The figure shows the latency of a server, installed at the UT, which provides a public mirror for a popular Linux distribution. The median latency users experienced when retrieving a file from the server is shown for a 6-hour interval in 1-minute time bins (note the logarithmic y-scale). As we can observe, the median latency is slightly higher than 200 ms most of the time, except for some particular intervals where a sharp increase can be observed. These surges in latency happen because the server is configured to synchronize with upstream repositories periodically, impacting the performance for end-users. System administrators can therefore rely on similar monitoring setups for analyzing application performance, even if servers are not under their control and, moreover, without instrumenting client devices.

The two examples discussed in this section illustrate the main pros and cons designers of flow-based performance monitoring applications are facing. An approach solely relying on common IEs comes with very low deployment costs, as it is likely compatible with an existing flow monitoring infrastructure. On the other hand, performance metrics such as one-way delay are not directly supported by many flow exporters and may require substantial calibration and compensation efforts. In contrast, enterprise-specific IEs allow to implement sophisticated monitoring techniques, but they rely heavily on DPI, resulting in higher deployment and privacy costs. We refer to [88], [89], [90] for further reading on flow-based performance monitoring.

D. Open-Source Tools & Commercial Appliances

When selecting a flow data analysis tool for deployment, regardless of whether an open-source or commercial one is selected, it is important to verify the following criteria:

- *Performance* – The performance of data analysis applications is usually measured by means of interface responsiveness; well-performing applications provide traffic reports on-the-fly, for an arbitrary number of data sources. The performance also depends on the amount of data to be processed, which depends on how historical data is handled, e.g., how long raw and aggregated data is stored.
- *Integration with systems and data sources* – Rather than solely relying on flow data, data analysis usually benefits from integrating with other data sources, such as geolocation databases, WHOIS, blacklists, BGP information, etc. In terms of system integration, support for directory services for user authentication are a welcome feature.
- *Analysis delay* – This delay should not be confused with the *processing delay* of a flow collector; while the processing delay determines when flow data is made available for analysis, the analysis delay is based on the computation time of the analysis software. The shorter the computation time, the more timely the analysis. Especially for time-critical applications, such as IDSs, analysis delays are an important criterium.

The market of commercial flow analysis applications consists of both appliance products (hardware or virtual) and software (standalone or *Software as a Service*). An overview of applications that have a primary focus on flow data analysis, is provided in Table VIII. Several conclusions can be drawn from this table. First, all applications provide Flow Analysis & Reporting functionality, usually complemented with Threat Detection or Performance Monitoring functionality. Very few applications provide both threat detection and performance monitoring functionality. Second, those applications doing performance monitoring have a strong focus on application performance, which is in line with the observation that application awareness in flow monitoring is becoming more important.

The number of available open-source flow data analysis applications that have been updated at least once since 2008 is rather small. We have compiled an overview in Table IX. Contrary to commercial applications, open-source alternatives are usually rather limited in functionality; although they all support flow analysis & reporting, extended functionality like performance monitoring is rare. Moreover, threat detection functionality is not supported by any open-source application. Some applications, such as NfSen, do however provide plugin-support, by means of which threat detection or performance monitoring can be implemented.

As flow data consists of large volumes of essentially tabular, timestamped information that is not very semantically complex, existing work in data analysis for other fields may prove to be applicable to flow data as well. We have had success in using the open-source *pandas*[18] data analysis framework for Python, together with glue code for bridging the gap between IPFIX files and *pandas* data-frames [93]. *Pandas* was originally developed for financial analysis and visualization, and is based on the *numpy* numerical computing framework for Python, which provides efficient primitives for dealing

[18]http://pandas.pydata.org/

TABLE VIII
COMMERCIAL FLOW DATA ANALYSIS APPLICATIONS

Vendor	Product	Flow analysis & reporting	Threat detection	Performance monitoring	Main selling point(s)
Arbor Networks	Pravail Network Security Intelligence (NSI)	✓	✓		Global threat intelligence, DDoS attack detection & mitigation
Compuware	Compuware APM	(✓)[1]		✓	Application-aware network monitoring (Quality of Experience)
Fluke Networks	Visual TruView	✓		✓	Application, network and VoIP performance monitoring
IBM	QRadar, QFlow	✓	✓		Security event correlation and NetFlow-based network behavior analysis
InfoVista	5View NetFlow	✓		✓	Application-aware network monitoring and reporting
INVEA-TECH	FlowMon Collector + ADS	✓	✓	✓	Traffic monitoring, threat (anomaly) detection and performance monitoring
Lancope	StealthWatch FlowCollector	✓	✓	✓	Security and performance monitoring
ManageEngine	NetFlow Analyzer	✓	✓		Traffic visibility and anomaly detection
Plixer	Flow Analytics	✓	✓	✓	Traffic monitoring, threat detection and performance monitoring
Riverbed Technology	Cascade Gateway	✓	(✓)[1]	✓	Network and application performance management
SevOne	SevOne Performance Appliance Solution	✓		✓	Network and application performance management
SolarWinds	NetFlow Traffic Analyzer	✓			Network utilization and bandwidth monitoring

[1] This feature is partially supported and not the principal functionality of the application or appliance.

TABLE IX
OPEN-SOURCE FLOW DATA ANALYSIS APPLICATIONS

Name	Flow analysis & reporting	Performance monitoring
FlowViewer	✓	
NfSen [73]	✓	
ntop/ntopng [91]	✓	
SiLK [74]	✓	
Stager [92]	✓	
WebView NetFlow Reporter	✓	✓

with large in-memory arrays of numeric data. It provides an environment for rapid exploratory analysis of relatively small datasets, crucial when designing and testing new algorithms and approaches for data analysis. Together with *matplotlib* and the "notebook" feature of *iPython*, it also provides a method for publishing analysis code work-in-progress along with data for collaborative and teaching tasks. Work in this areas is ongoing.

VIII. LESSONS LEARNED – COMMON PITFALLS

In the previous sections, we have discussed how to setup a typical flow monitoring system. Before the exported flow data can however be used for production or measurement purposes, the setup has to be verified and calibrated. This section will discuss how hidden problems that impact the resulting data can be detected and potentially overcome. We start by describing flow exporter overload in Section VIII-A, followed by transport overload and flow collector overload in Section VIII-B and VIII-C, respectively. Finally, we discuss several flow data artifacts that can be found on several flow export devices in Section VIII-D.

A. Flow Exporter Overload

Flow caches in a flow exporter usually have a fixed size that is either constrained by hardware, or determined at compile-time in the case of flow exporter software. When this size turns out to be small, flow data loss or low performance can be the result. The latter is especially the case for software-based solutions, as they often use linked lists to store different flow cache entries under the same hash value, which results in longer cache entry lookup times. Given that it is not always possible to foresee significant changes in the monitored traffic, flow caches may eventually turn out under-dimensioned.

Since under-dimensioned flow caches usually result in data loss it is important to be aware of this happening, especially when the exported flow data is used for critical applications. For those having access to a flow exporter, be it a packet forwarding device or dedicated probe, it is usually trivial to obtain these data loss statistics. For example, software exporters often write them to log files, while the flow cache utilization and loss statistics of hardware flow exporters can be obtained via a command-line interface (CLI), or SNMP. An example of how to retrieve such details from Cisco switches is provided in [64]. For those having only access to the exported flow data, it is much harder to derive conclusions about data loss. An example of this is also provided in [64], where it is shown that indications of an under-dimensioned flow cache can be retrieved from the dataset with some uncertainty.

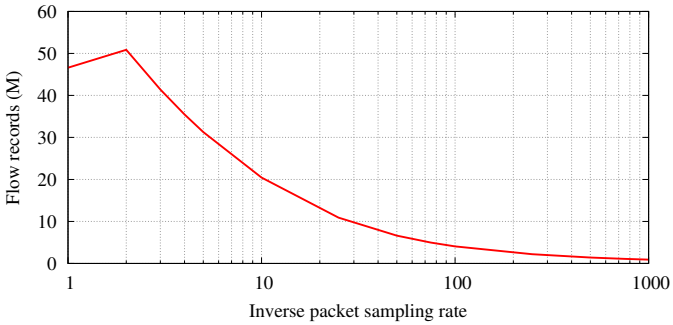


Fig. 16. Impact of packet sampling on the number of flow records.

Several actions can be taken to reduce flow exporter load, without the need to replace a flow exporter. First, expiration timeouts can be reduced. Especially the idle timeout should be considered here, as it expires cache entries of flows that are inactive anyway. Although reducing timeouts results in a lower flow cache utilization, which has been shown in Section V-C, this will result in more flow records. Care should be taken to not overload a flow exporter's Exporting Process or a flow collector with this larger number of flow records.

A second action for reducing the load of flow exporters is enabling packet sampling or decreasing the packet sampling rate, i.e., reducing the number of packets forwarded to the Metering Process. We have measured the impact of packet sampling on the resulting flow data based on the dataset presented in Section I-B. The results are shown in Fig. 16 and two conclusions can be derived from this figure. First, the use of packet sampling does not necessarily reduce the number of flow records, mostly because of the use of timeout-based expiration and the nature of the traffic; when intermediate packets in a flow are not sampled and therefore not considered by the Metering Process, this flow can be split into multiple flows by the applied idle timeout. This demonstrates that packet sampling reduces the load of the Metering Process, but not necessarily of the subsequent stages. Second, the impact of packet sampling on the number of flow records reduces when the sampling rate is increased (except for very high rates, such as 1:2, in our dataset). In contrast to packet sampling, flow sampling reduces the number of flow records, but it will not help to reduce utilization of the flow cache, as it is applied after the Metering Process. Enabling packet sampling or increasing its rate however results in information loss that is in various cases very hard to (mathematically) compensate for. A flow-based anomaly detection system, for example, which is based on thresholds determined on a non-sampled dataset or datasets captured using another sampling rate, will function sub-optimally or stop functioning completely. It is therefore advised to tune flow entry expiration first, before enabling or modifying sampling parameters.

As soon as a flow exporter is experiencing capacity problems due to resource constraints, flow records may start to be expired in a different way. That means, the active and idle timeouts are respected as much as possible, until the utilization of the flow cache exceeds a threshold (e.g., 90%). At that moment, an emergency expiration kicks in which expires cache entries prematurely, to free up the flow cache and to allow new entries to be inserted. This results in more flow records, and flow data that is not expired consistently, which may impact the subsequent Data Analysis stage. For example, an intrusion detection system (IDS) that counts the number of flow records for detecting a particular anomaly may not function properly anymore by raising false alerts. It is therefore important to be aware of these dynamics in flow export setups.

TABLE X
EXPORT VOLUMES FOR THE UT DATASET (2.1 TB)

Sampling rate	Protocol	Export packets / bytes
1:1	NetFlow v5	1.4 M / 2.1 G
1:1	NetFlow v9	3.5 M / 2.5 G
1:10		1.6 M / 1.1 G
1:100		314.9 k / 222.5 M
1:1000		72.2 k / 49.5 M
1:1	IPFIX	4.3 M / 3.0 G

B. Transport Overload

It is not uncommon for flow exporters to export data from links of 10 Gbps and higher over links of 1 Gbps. This is usually the case because flow data is sent to collectors of the exporter's management interface (which usually has a line speed of 1 Gbps), and because flow collectors are often "normal" file servers that are not equipped with special, high-speed network interfaces. Due to the data reduction achieved by flow export, flow data exported from high-speed links can generally be exported over smaller links without data loss. However, in anomalous situations as described in the previous subsection, such links will become a bottleneck in a flow monitoring system. This is especially the case in anomalous situations, where the data aggregation achieved by flow export is constrained, such as under DDoS attacks. One type of DDoS attacks are flooding attacks, which aim at overloading targets by opening many new connections by sending a large number of TCP `SYN`-packets, for example. Due to the definition of a flow, this type of traffic results in many new flows, because of the changing flow key in every packet [94]. Moreover, depending on the selected number of IEs for each flow, the data aggregation usually achieved can change into data export that is neutral in size (i.e. the exported data has the same size as the monitored data) or even data amplification; as soon as the overhead of the IPFIX Message is larger than the monitored packets on the line (e.g., during a flooding attack), amplification takes place. This is not uncommon, as many flow exporters use by default a set of IEs that is larger in terms of bytes than a TCP `SYN`-packet, for example.

We have measured the volume of NetFlow and IPFIX Messages in terms of packet and bytes for the UT dataset presented in Section I-B. The results for various packet sampling rates and export protocols are shown in Table X. Two main observations can be made. First, the newer the flow export protocol, the more packets and bytes are sent to the flow collector. This can be explained by the inclusion of (option) templates in NetFlow v9 and IPFIX, and the

increase in timestamp size from 32 bits relative timestamps (in NetFlow v9) to 64 bits absolute timestamps[19] in IPFIX. Second, regardless of the export protocol used, NetFlow and IPFIX traffic is roughly 0.1% of the original traffic in a setup without packet sampling. This is in contrast to [55], where it is claimed that the IPFIX traffic generated by a flow exporter is 2-5% of the traffic of the monitored link.

Both NetFlow and IPFIX carry sequence numbers in their respective packet headers, which assist in identifying packet loss when either unreliable transports are used (e.g., UDP) or the transport bandwidth is permanently under-dimensioned. Flow collectors, such as *nfdump*, keep track of these sequence numbers and store the number of sequence failures in the metadata of each file. Before deriving any conclusion from sequence failure counters, attention should be paid to the export protocol version. Both NetFlow v5 and IPFIX provide sequence numbers in terms of flow records, while NetFlow v9 provides these in terms of export packets. As such, the actual number of missing flow records can only be estimated when NetFlow v5 or IPFIX are used.

C. Flow Collector Overload

Flow data collection usually gets least attention of all stages in a typical flow monitoring setup, although a suboptimal setup can result in severe – and often unnoticed – data loss. What makes it difficult to dimension a flow collector, is that in anomalous situations, the number of incoming flow records can be doubled or even more than that. Considerable over-provisioning and calibration are therefore needed to ensure that no data is lost. In this subsection, we discuss how flow collectors should be calibrated by means of performance measurements.

Data loss as part of flow collector overload can be a consequence of both kernel and application buffer overflows, and disk I/O overload. Kernel buffers store NetFlow and IPFIX Messages before they are received by a Collecting Process. Application buffers are part of the flow collector itself and can be used for any intermediate processing before the flow data is stored. Buffers can be tuned to a certain extent; increasing buffer sizes allows more data to be temporarily stored, but is useless if subsequent system elements are becoming a bottleneck. In practice, disk I/O will be a bottleneck in such situations, so increasing buffers provides only limited advantages.

Many flow collectors apply data compression to flow data by default. Whether or not compression should be enabled depends on the processing and storage capacities of the system acting as a flow collector. As a rule of thumb, one can compare the time needed to store a flow data chunk both in compressed and uncompressed format. If writing compressed data is faster, the storage subsystem is the bottleneck of the collection system and data compression, which is CPU-intensive, should be enabled. Otherwise, if writing uncompressed data is faster, processing capacity is the bottleneck and data compression should be disabled. This rule of thumb is confirmed by *nftest*, part of *nfdump*, which performs these tests automatically and provides one with an advice on whether or not to enable data compression.

[19]IPFIX supports timestamps at multiple granularities, ranging from seconds to nanoseconds. The used flow exporter, *nProbe*, uses millisecond resolution timestamps by default for IPFIX.

TABLE XI
STORAGE VOLUMES FOR THE UT DATASET (2.1 TB)

Sampling rate	Protocol	Storage volume	Reduction factor
1:1	NetFlow v5	912.7 MB	2301x
1:1	NetFlow v9	1.0 GB	2100x
1:10		503.7 MB	4169x
1:100		103.9 MB	20212x
1:1000		20.4 MB	102941x
1:1	IPFIX	820.4 MB	2560x

To get an idea of how much processing capacity is needed to store flow data of one day, we have performed several measurements on the UT dataset presented in Section I-B, where we have used *nfdump* as the flow collector software. The storage volumes for various export parameters are listed in Table XI. The listed storage volumes are for compressed datasets and are slightly less than 1 GB per day when no packet sampling is used. To put this in contrast; the Czech National Research and Education Network (NREN) CESNET does not apply packet sampling either and stores roughly 125 GB of flow data per day, while SURFnet, the Dutch NREN, stores around 16 GB per day with a packet sampling rate of 1:100.

D. Flow Data Artifacts

Analyses have shown that the advantages offered by flow export often come at the expense of accuracy, although the gains of using flow data normally excuse this. Since IPFIX is still in its early days and NetFlow deployment is far more mature [3], most literature on flow data artifacts is about NetFlow (especially v9).

Flow data artifacts described in literature can be classified into three categories:

- Timing, related to the way in which flow exporters put timestamps into flow records, how these timestamps are stored by export protocols, and how precise flow exporters are in expiring flow records.
- Data loss, causing unrepairable damage to flow datasets.
- Minor inaccuracies that can usually be repaired or ignored.

Artifacts in the first category, timing, are all related to the way in which NetFlow accounts flow record start and end times. NetFlow v9 in particular uses two separate clocks: an uptime clock in milliseconds that is used to express flow record start and end times in terms of a flow exporter's uptime, and a real-time clock (UNIX time) that is used to map those uptimes to an absolute time. The real time is inserted in NetFlow packets together with the uptime before they are transmitted to a flow collector. It is then up to a flow collector to calculate the absolute start and end times of flow records based on these two types of timestamps. The advantage of using only a single real-time clock is that there are no two clocks that

need to remain synchronized all the times. However, several artifacts related to timing have been reported in literature. First, it is explained in [90] and [95] that the millisecond-level precision of the flow exporter uptimes is sacrificed, since only second-level timestamps of the real time can be stored in a NetFlow v9 packet. This leads to imprecise or incorrect start and end times of flow records. The same works describe that both clocks are not necessarily synchronized, resulting in a clock skew in the order of seconds per day. In addition, two timestamps are not necessarily inserted into the NetFlow packet at exactly the same moment either due to resource exhaustion or explicit export rate limiting, resulting in an additional delay [95]. Another category of timing artifacts is the imprecise or erroneous expiration of flow records, resulting in periodic patterns in the flow dataset and erroneously merged flow records [64], [96].

The second category of artifacts is related to data loss. It is described in [64] that many older yet widespread Cisco routers and switches – Cisco's Catalyst 6500 in particular, which has been Cisco's most-sold device – do not export TCP flags for the majority of flows. Since TCP flags are useful to infer TCP connection states, many analysis applications will have to disable part of their functionality because of this missing information. Another artifact related to data loss described in [64] and [96] is related to gaps in flow data, usually caused by an under-dimensioned flow cache, as described in Section VIII-A. As a consequence, packets cannot always be accounted to flow records in the flow cache, effectively resulting in data loss.

Two artifacts causing minor inaccuracies in flow data are also described in [64]. First, invalid byte counters in flow records can be caused by certain Cisco routers in the case of very small Ethernet frames. This happens because the padding bytes in Ethernet frames are not correctly stripped in those situations. Second, some flow records exported by older Cisco routers have TCP flags set for non-TCP flows. This is, however, not problematic for the flow data itself and can usually be ignored.

For many application areas, such as traffic accounting, the artifacts described in this section do not play a major role. However, as soon as the flow data is used for research purposes, e.g., for flow-based delay measurements [90], more attention should be paid. Flow exporters must be calibrated before their data is used for such purposes. Also analysis applications have to be verified whether they will work as expected with flow data from a certain flow exporter. Interoperability tests are organized from time to time to test protocol compliance of implementations from various vendors. This is however only on the level of protocols and not on the level of flow records. Up to a certain degree, the variance and imprecision of flow exporters when exporting flows is even tolerated by the specifications. For example, for flow cache entry expiration, as described in Section V-C, a certain degree of freedom is left for flow exporters in case of resource constraints.

IX. Concluding Remarks & Outlook

This tutorial has shown and discussed all aspects of a full-fledged flow monitoring setup based on NetFlow or IPFIX, covering the complete spectrum of packet observation, flow metering and export, data collection, and data analysis. The generic architecture of such a setup has been explained in Section III. It has been shown that each of these stages affects the final flow data and consequently, its analysis. Understanding all these stages to avoid measurement artifacts is therefore of key importance to anyone performing flow measurements, as demonstrated in Section VIII.

One of the most prevalent trends in flow export is certainly the flexibility with respect to which data is exported. This is clearly shown by IPFIX, which allows one to tailor virtually anything to the needs of data analysis, as shown in Section V. In contrast to what the name suggests, IPFIX can be used to export any traffic information from L2-L7. A concrete example of this has been discussed extensively in Section V of this paper: application awareness.

The applicability of IPFIX can even be taken to the next level by exploiting its flexibility and extending the set of IEs beyond network traffic information. As long as flow exporters know how to insert measurement data into IPFIX Messages, existing flow monitoring infrastructure and applications can be used, such as flow collectors, and analysis software. A proof-of-concept has been demonstrated as part of an IETF tutorial on IPFIX [97], where the room temperature was exported to a flow collector using IPFIX. Existing software could be used for monitoring the temperature over time and configuring thresholds and alerts. Another untypical use case is the transport of syslog or SNMP data using IPFIX, which are features recently offered by several vendors and discussed within the IETF, respectively.

Given the mentioned developments, we consider IPFIX – in contrast to what the name suggests – a generic transport protocol for structured data, rather than a pure flow export protocol. More applications based on IPFIX that go beyond flow export will certainly follow, a typical example being the Internet of Things (IoT), where a plethora of sensor devices is connected to the Internet. Since these devices produce structured measurement data that has to be collected and analyzed, we believe that these new application areas can definitely utilize the principles developed and work performed by the flow measurement community.

Acknowledgment

The authors would like to thank Cyndi Mills, Benoit Claise (Cisco Systems Inc.) and Nevil Brownlee (University of Auckland) for their valuable contributions.

This work was partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

REFERENCES

[1] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011 (Internet Standard), Internet Engineering Task Force, September 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7011.txt

[2] N. Brownlee, "Flow-Based Measurement: IPFIX Development and Deployment," *IEICE Transactions on Communications*, vol. 94, no. 8, pp. 2190–2198, 2011.

[3] J. Steinberger, L. Schehlmann, S. Abt, and H. Baier, "Anomaly Detection and mitigation at Internet scale: A survey," in *Proceedings of the 7th International Conference on Autonomous Infrastructure, Management and Security, AIMS'13*, ser. Lecture Notes in Computer Science, vol. 7943. Springer Berlin Heidelberg, 2013, pp. 49–60.

[4] European Parliament & Council, "Directive 2006/24/EC of the European Parliament and of the Council of 15 March 2006 on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks and amending Directive 2002/58/EC," March 2006, accessed on 2 May 2014. [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2006:105:0054:0063:EN:PDF

[5] W. John, S. Tafvelin, and T. Olovsson, "Passive Internet Measurement: Overview and Guidelines based on Experiences," *Computer Communications*, vol. 33, no. 5, pp. 533–550, 2010.

[6] C. Mills, D. Hirsh, and G. Ruth, "Internet Accounting: Background," RFC 1272, Internet Engineering Task Force, November 1991. [Online]. Available: http://www.ietf.org/rfc/rfc1272.txt

[7] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "A Parameterizable Methodology for Internet Traffic Flow Profiling," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1481–1494, 1995.

[8] N. Brownlee, "RTFM: Applicability Statement," RFC 2721 (Informational), Internet Engineering Task Force, October 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2721.txt

[9] Cisco Systems, Inc., "Cisco Catalyst 6500 Architecture White Paper," 2013, accessed on 2 May 2014. [Online]. Available: http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps708/prod_white_paper0900aecd80673385.html

[10] ——, "Cisco IOS NetFlow and Security," February 2005, accessed on 2 May 2014. [Online]. Available: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6586/ps6642/prod_presentation0900aecd80311f49.pdf

[11] ——, "NetFlow Services Solutions Guide," 2007, accessed on 2 May 2014. [Online]. Available: http://www.cisco.com/en/US/docs/ios/solutions_docs/netflow/nfwhite.html

[12] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954 (Informational), Internet Engineering Task Force, October 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[13] Cisco Systems, Inc., "Cisco IOS Flexible NetFlow," December 2008, accessed on 2 May 2014. [Online]. Available: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/ps6965/product_data_sheet0900aecd804b590b.html

[14] L. Deri, E. Chou, Z. Cherian, K. Karmarkar, and M. Patterson, "Increasing Data Center Network Visibility with Cisco NetFlow-Lite," in *Proceedings of the 7th International Conference on Network and Service Management, CNSM'11*, 2011, pp. 1–6.

[15] IETF, "IP Flow Information Export (ipfix)," accessed on 2 May 2014. [Online]. Available: http://datatracker.ietf.org/wg/ipfix/charter/

[16] J. Quittek, T. Zseby, B. Claise, and S. Zander, "Requirements for IP Flow Information Export," RFC 3917 (Informational), Internet Engineering Task Force, October 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3917.txt

[17] S. Leinen, "Evaluation of Candidate Protocols for IP Flow Information Export," RFC 3955 (Informational), Internet Engineering Task Force, October 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3955.txt

[18] B. Trammell and E. Boschi, "An Introduction to IP Flow Information Export (IPFIX)," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 89–95, 2011.

[19] P. Phaal, "sFlow Version 5," July 2004, accessed on 2 May 2014. [Online]. Available: http://sflow.org/sflow_version_5.txt

[20] B. Claise, A. Johnson, and J. Quittek, "Packet Sampling (PSAMP) Protocol Specifications," RFC 5476 (Proposed Standard), Internet Engineering Task Force, March 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5476.txt

[21] Open Networking Foundation, "OpenFlow Switch Specification," September 2012, accessed on 2 May 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf

[22] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[23] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *Proceedings of the 14th International Conference on Passive and Active Measurement, PAM'13*, ser. Lecture Notes in Computer Science, vol. 7799. Springer Berlin Heidelberg, 2013, pp. 31–41.

[24] S. Seetharaman, "OpenFlow/SDN tutorial OFC/NFOEC," in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2012 and the National Fiber Optic Engineers Conference*, 2012, pp. 1–52.

[25] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, March 2008.

[26] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek, "Architecture for IP Flow Information Export," RFC 5470 (Informational), Internet Engineering Task Force, March 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5470.txt

[27] A.-C. Orgerie, P. Gonçalves, M. Imbert, J. Ridoux, and D. Veitch, "Survey of Network Metrology Platforms," in *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet, SAINT'12*, 2012, pp. 220–225.

[28] J. Micheel, S. Donnelly, and I. Graham, "Precision Timestamping of Network Packets," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, IMW'01*, 2001, pp. 273–277.

[29] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall, "Sampling and Filtering Techniques for IP Packet Selection," RFC 5475 (Proposed Standard), Internet Engineering Task Force, March 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5475.txt

[30] J. Weber, "The Fundamentals of Passive Monitoring Access," June 2006, accessed on 2 May 2014. [Online]. Available: http://www.nanog.org/meetings/nanog37/presentations/joy-weber.pdf

[31] Cisco Systems, Inc., "Switched Port Analyzer (SPAN)," 2007, accessed on 2 May 2014. [Online]. Available: http://www.cisco.com/en/US/tech/tk389/tk816/tk834/tsd_technology_support_sub-protocol_home.html

[32] J. Zhang and A. Moore, "Traffic trace artifacts due to monitoring via port mirroring," in *Proceedings of the 15th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services, E2EMON'07*, 2007, pp. 1–8.

[33] A. Orebaugh, G. Ramirez, J. Burke, L. Pesce, J. Wright, and G. Morris, "Chapter 6 – Wireless Sniffing with Wireshark," in *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress, 2013, pp. 267–370.

[34] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending Networking into the Virtualization Layer," in *8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[35] S. Alcock, P. Lorier, and R. Nelson, "Libtrace: A Packet Capture and Analysis Library," *SIGCOMM Computer Communication Review*, vol. 42, no. 2, pp. 42–48, 2012.

[36] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle, "Comparing and Improving Current Packet Capturing Solutions based on Commodity Hardware," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, IMC'10*, 2010, pp. 206–217.

[37] L. Degioanni and G. Varenni, "Introducing Scalability in Network Measurement: Toward 10 Gbps with Commodity Hardware," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, IMC'04*, 2004, pp. 233–238.

[38] J. L. García-Dorado, F. Mata, J. Ramos, P. M. S. del Río, V. Moreno, and J. Aracil, "High-Performance Network Traffic Processing Systems using Commodity Hardware," in *Data Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science, 2013, vol. 7754, pp. 3–27.

[39] F. Fusco and L. Deri, "High Speed Network Traffic Analysis with Commodity Multi-core Systems," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, IMC'10*, 2010, pp. 218–224.

[40] P. D. Amer and L. N. Cassel, "Management of sampled real-time network measurements," in *Proceedings of the 14th Conference on Local Computer Networks*, 1989, pp. 62–68.

[41] N. Duffield, "Sampling for Passive Internet Measurement: A Review," *Statistical Science*, vol. 19, no. 3, pp. 472–498, 2004.

[42] N. Duffield, C. Lund, and M. Thorup, "Estimating Flow Distributions from Sampled Flow Statistics," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 933–946, 2005.

[43] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a Better NetFlow," in *Proceedings of the ACM SIGCOMM 2004 Conference, SIGCOMM'04*, 2004, pp. 245–256.

[44] IANA, "IP Flow Information Export (IPFIX) Entities," June 2013, accessed on 2 May 2014. [Online]. Available: http://www.iana.org/assignments/ipfix/ipfix.xml

[45] B. Claise and B. Trammell, "Information Model for IP Flow Information Export," RFC 7012 (Standards Track), Internet Engineering Task Force, September 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7012.txt

[46] R. Hofstede, I. Drago, A. Sperotto, and A. Pras, "Flow Monitoring Experiences at the Ethernet-Layer," in *Energy-Aware Communications. Proceedings of the 17th EUNICE Open European Summer School, EUNICE'11*, ser. Lecture Notes in Computer Science, vol. 6955. Springer Berlin Heidelberg, 2011, pp. 134–145.

[47] P. Aitken, B. Claise, S. B.S., C. McDowall, and J. Schönwälder, "Exporting MIB Variables using the IPFIX Protocol," Internet Engineering Task Force, March 2014. [Online]. Available: http://www.ietf.org/id/draft-ietf-ipfix-mib-variable-export-05.txt

[48] B. Trammell and B. Claise, "Guidelines for Authors and Reviewers of IP Flow Information Export (IPFIX) Information Elements," RFC 7013 (Best Current Practice), Internet Engineering Task Force, September 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7013.txt

[49] I. Drago, R. R. Barbosa, R. Sadre, A. Pras, and J. Schönwälder, "Report of the Second Workshop on the Usage of NetFlow/IPFIX in Network Management," *Journal of Network and Systems Management*, vol. 19, no. 2, pp. 298–304, 2011.

[50] M. Patterson, "NetFlow v9 vs. NetFlow v5: What Are the Differences?" June 2009, accessed on 2 May 2014. [Online]. Available: http://www.plixer.com/blog/netflow/netflow-v9-vs-netflow-v5/

[51] B. Claise, G. Dhandapani, P. Aitken, and S. Yates, "Export of Structured Data in IP Flow Information Export (IPFIX)," RFC 6313 (Standards Track), Internet Engineering Task Force, July 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6313.txt

[52] B. Trammell and E. Boschi, "Bidirectional Flow Export Using IP Flow Information Export (IPFIX)," RFC 5103 (Standards Track), Internet Engineering Task Force, January 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5103.txt

[53] Cisco Systems, Inc., "Cisco IOS Flexible NetFlow Commands," December 2010, accessed on 2 May 2014. [Online]. Available: http://www.cisco.com/en/US/docs/ios/fnetflow/command/reference/fnf_01.pdf

[54] S. D'Antonio, T. Zseby, C. Henke, and L. Peluso, "Flow Selection Techniques," RFC 7014 (Standards Track), Internet Engineering Task Force, September 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7014.txt

[55] E. Boschi, L. Mark, J. Quittek, M. Stiemerling, and P. Aitken, "IP Flow Information Export (IPFIX) Implementation Guidelines," RFC 5153 (Informational), Internet Engineering Task Force, April 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5153.txt

[56] R. R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Standards Track), Internet Engineering Task Force, September 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt

[57] B. Claise, P. Aitken, A. Johnson, and G. Münz, "IP Flow Information Export (IPFIX) Per Stream Control Transmission Protocol (SCTP) Stream," RFC 6526 (Standards Track), Internet Engineering Task Force, March 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6526.txt

[58] R. R. Stewart, M. A. Ramalho, Q. Xie, M. Tuexen, and P. T. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension," RFC 3758 (Standards Track), Internet Engineering Task Force, May 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3758.txt

[59] B. Trammell, E. Boschi, L. Mark, T. Zseby, and A. Wagner, "Specification of the IP Flow Information Export File Format," RFC 5655 (Proposed Standard), Internet Engineering Task Force, October 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5655.txt

[60] L. Deri, "nProbe: An Open Source NetFlow Probe for Gigabit Networks," in *Proceedings of the TERENA Networking Conference, TNC'03*, 2003.

[61] P. Lucente, "pmacct: steps forward interface counters," Tech. Rep., 2008. [Online]. Available: http://www.pmacct.net/pmacct-stepsforward.pdf

[62] R. T. Lampert, C. Sommer, G. Munz, and F. Dressler, "Vermont – A Versatile Monitoring Toolkit for IPFIX and PSAMP," in *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation, MonAM'06*, 2006.

[63] C. M. Inacio and B. Trammell, "YAF: Yet Another Flowmeter," in *Proceedings of the 24th International Conference on Large Installation System Administration, LISA'10*, 2010, pp. 1–16.

[64] R. Hofstede, I. Drago, A. Sperotto, R. Sadre, and A. Pras, "Measurement Artifacts in NetFlow Data," in *Proceedings of the 14th International Conference on Passive and Active Measurement, PAM'13*, ser. Lecture Notes in Computer Science, vol. 7799. Springer Berlin Heidelberg, 2013, pp. 1–10.

[65] A. Finamore, M. Mellia, M. Meo, M. M. Munafò, and D. Rossi, "Experiences of Internet Traffic Monitoring with Tstat," *IEEE Network*, vol. 25, no. 3, pp. 8–14, 2011.

[66] F. Fusco, M. Vlachos, and X. Dimitropoulos, "RasterZip: compressing network monitoring data with support for partial decompression," in *Proceedings of the 2012 ACM conference on Internet Measurement Conference, IMC'12*, 2012, pp. 51–64.

[67] B. Trammell, A. Wagner, and B. Claise, "Flow Aggregation for the IP Flow Information Export Protocol," RFC 7015 (Standards Track), Internet Engineering Task Force, September 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7015.txt

[68] L. Deri, V. Lorenzetti, and S. Mortimer, "Collection and Exploration of Large Data Monitoring Sets Using Bitmap Databases," in *Traffic Monitoring and Analysis, TMA'10*, ser. Lecture Notes in Computer Science, vol. 6003. Springer Berlin Heidelberg, 2010, pp. 73–86.

[69] R. Hofstede, A. Sperotto, T. Fioreze, and A. Pras, "The Network Data Handling War: MySQL vs NfDump," in *Networked Services and Applications – Engineering, Control and Management. Proceedings of the 16th EUNICE Open European Summer School, EUNICE'10*, ser. Lecture Notes in Computer Science, vol. 6164. Springer Berlin Heidelberg, 2010, pp. 167–176.

[70] P. Velan, "Practical Experience with IPFIX Flow Collectors," in *Proceedings of the 13th IFIP/IEEE International Symposium on Integrated Network Management, IM'13*, 2013, pp. 1015–1020.

[71] C. Morariu, P. Racz, and B. Stiller, "SCRIPT: A Framework for Scalable Real-time IP Flow Record Analysis," in *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium, NOMS'10*, 2010, pp. 278–285.

[72] P. Velan and R. Krejčí, "Flow Information Storage Assessment Using IPFIXcol," in *Dependable Networks and Services. Proceedings of the 6th International Conference on Autonomous Infrastructure, Management and Security, AIMS'12*, ser. Lecture Notes in Computer Science, vol. 7279. Springer Berlin Heidelberg, 2012, pp. 155–158.

[73] P. Haag, "Watch your Flows with NfSen and NFDUMP," in *50th RIPE Meeting*, 2005. [Online]. Available: http://meetings.ripe.net/ripe-50/presentations/ripe50-plenary-tue-nfsen-nfdump.pdf

[74] C. Gates, M. Collins, M. Duggan, A. Kompanek, and M. Thomas, "More Netflow Tools for Performance and Security," in *Proceedings of the 18th International Conference on Large Installation System Administration, LISA'04*, 2004, pp. 121–132.

[75] Cooperative Association for Internet Data Analysis (CAIDA), "Summary of Anonymization Best Practice Techniques," accessed on 2 May 2014. [Online]. Available: http://www.caida.org/projects/predict/anonymization/

[76] C. Schmoll, S. Teofili, E. Delzeri, G. Bianchi, I. Gojmerac, E. Hyytia, B. Trammell, E. Boschi, G. Lioudakis, F. Gogoulos, A. Antonakopoulou, D. Kaklamani, and I. Venieris, "State of the art on data protection algorithms for monitoring systems: FP7-PRISM IST-2007-215350 Deliverable 3.1.1," June 2008, accessed on 2 May 2014. [Online]. Available: http://fp7-prism.eu/images/upload/Deliverables/fp7-prism-wp3.1-d3.1.1-final.pdf

[77] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon, "Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme," *Computer Networks*, vol. 46, no. 2, pp. 253–272, 2004.

[78] M. Burkhart, D. Schatzmann, B. Trammell, E. Boschi, and B. Plattner, "The Role of Network Trace Anonymization under Attack," *SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 5–11, 2010.

[79] T. Zseby, E. Boschi, N. Brownlee, and B. Claise, "IP Flow Information Export (IPFIX) Applicability," RFC 5472 (Informational), Internet Engineering Task Force, March 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5472.txt

[80] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A Survey of Network Flow Applications," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 567–581, 2013.

[81] P. Haag, "Tracking Incidents with NfSen," 2008, accessed on 2 May 2014. [Online]. Available: http://www.switch.ch/export/sites/default/all/cert/downloads/presentations/_files_presentations/Tracking.pdf

[82] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras, "Towards Real-Time Intrusion Detection for NetFlow/IPFIX," in *Proceedings of the 9th International Conference on Network and Service Management, CNSM'13*, 2013, pp. 227–234.
[83] R. Hofstede and T. Fioreze, "SURFmap: A Network Monitoring Tool Based on the Google Maps API," in *Application Session Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management, IM'09*, 2009, pp. 676–690.
[84] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An Overview of IP Flow-Based Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.
[85] Mandiant, "APT1 – Exposing One of China's Cyber Espionage Units," February 2013, accessed on 2 May 2014. [Online]. Available: http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf
[86] A. Sperotto, R. Sadre, P.-T. de Boer, and A. Pras, "Hidden Markov Model modeling of SSH brute-force attacks," in *Integrated Management of Systems, Services, Processes and People in IT. Proceedings of the 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM'09*, ser. Lecture Notes in Computer Science, vol. 5841. Springer Berlin Heidelberg, 2009, pp. 164–176.
[87] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "SSHCure: A Flow-Based SSH Intrusion Detection System," in *Dependable Networks and Services. Proceedings of the 6th International Conference on Autonomous Infrastructure, Management and Security, AIMS'12*, ser. Lecture Notes in Computer Science, vol. 7279. Springer Berlin Heidelberg, 2012, pp. 86–97.
[88] L. Deri, "How to Monitor Latency Using nProbe," 2011, accessed on 2 May 2014. [Online]. Available: http://www.ntop.org/nprobe/how-to-monitor-latency-using-nprobenagios-world-conference-europe/
[89] D. Schatzmann, S. Leinen, J. Kögel, and W. Mühlbauer, "FACT: Flow-Based Approach for Connectivity Tracking," in *Proceedings of the 12th International Conference on Passive and Active Network Measurement, PAM'11*, 2011, pp. 214–223.
[90] J. Kögel, "One-way Delay Measurement based on Flow Data: Quantification and Compensation of Errors by Exporter Profiling," in *Proceedings of the International Conference on Information Networking, ICOIN'11*, 2011, pp. 25–30.
[91] L. Deri and S. Suin, "Ntop: Beyond Ping and Traceroute," in *Active Technologies for Network and Service Management. Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM'99*, ser. Lecture Notes in Computer Science, vol. 1700. Springer Berlin Heidelberg, 1999, pp. 271–283.
[92] A. Øslebø, "Stager – A Web Based Application for Presenting Network Statistics," in *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium, NOMS'06*, 2006, pp. 1–156.
[93] B. Trammell, "Integrating IPFIX with Pandas for Exploratory Analysis in Research," July 2013, accessed on 2 May 2014. [Online]. Available: http://www.ietf.org/proceedings/87/slides/slides-87-nmrg-1.pdf
[94] R. Sadre, A. Sperotto, and A. Pras, "The Effects of DDoS Attacks on Flow Monitoring Applications," in *Proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium, NOMS'12*, 2012, pp. 269–277.
[95] B. Trammell, B. Tellenbach, D. Schatzmann, and M. Burkhart, "Peeling away Timing Error in NetFlow Data," in *Proceedings of the 12th International Conference on Passive and Active Measurement, PAM'11*, ser. Lecture Notes in Computer Science, vol. 6579. Springer Berlin Heidelberg, 2011, pp. 194–203.
[96] I. Cunha, F. Silveira, R. Oliveira, R. Teixeira, and C. Diot, "Uncovering Artifacts of Flow Measurement Tools," in *Proceedings of the 10th International Conference on Passive and Active Network Measurement, PAM'09*, ser. Lecture Notes in Computer Science, vol. 5448. Springer Berlin Heidelberg, 2009, pp. 187–196.
[97] B. Trammell and B. Claise, "Applying IPFIX to Network Measurement and Management," July 2013, accessed on 2 May 2014. [Online]. Available: http://www.ietf.org/edu/tutorials/ipfix-tutorial.pdf

APPENDIX
LIST OF ABBREVIATIONS

APT	Advanced Persistent Threat
ASIC	Application-Specific Integrated Circuit
BGP	Border Gateway Protocol
CLI	Command-Line Interface
CPU	Central Processing Unit
DBMS	Database Management System
DMA	Direct Memory Access
DNS	Domain Name System
DDoS	Distributed DoS
DoS	Denial-of-Service
DPI	Deep Packet Inspection
DTLS	Datagram TLS
FPGA	Field-Programmable Gate Array
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
IDS	Intrusion Detection System
IE	Information Element
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPFIX	IP Flow Information eXport
LAN	Local Area Network
MAC	Medium Access Control
MIB	Management Information Base
MPLS	Multiprotocol Label Switching
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NFS	Network File System
NREN	National Research and Education Network
NTP	Network Time Protocol
PSAMP	Packet Sampling
RAID	Redundant Array of Independent Disks
REST	REpresentational State Transfer
RTFM	Realtime Traffic Flow Measurement
RTT	Round Trip Time
SCTP	Stream Control Transmission Protocol
SDN	Software-Defined Networking
SLA	Service-Level Agreement
SNMP	Simple Network Management Protocol
SNTP	Simple NTP
SPAN	Switched Port ANalyzer
SSD	Solid-State Drive
SSH	Secure SHell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual LAN
VM	Virtual Machine
VPN	Virtual Private Network
WG	(IETF) Working Group
WLAN	Wireless LAN

BIOGRAPHIES

Rick Hofstede is a Ph.D. candidate at the University of Twente, The Netherlands, where he graduated in Telematics in 2009 (B.Sc.) and in 2011 (M.Sc.). The working title of his thesis is "Real-Time and Resilient Intrusion Detection: A Flow-Based Approach". His main topics of interest are network security (intrusion detection and forensics in particular), Internet measurements and network data visualization.

Pavel Čeleda is a senior researcher affiliated with the Institute of Computer Science at the Masaryk University in Brno, Czech Republic. He received a Ph.D. degree in Informatics from University of Defence, Brno. His main research interests include flow monitoring, cyber security and design of network security devices. He participates in a number of academia, industry and defence projects. His research results were successfully transferred to three university startup companies.

Brian Trammell is a senior researcher at ETH Zurich's Communications Systems Group. He co-authored many of the IPFIX RFCs. He is a member of the Internet Architecture Board, and co-chairs the IETF's IP Performance Metrics Working Group. His current research interests include passive and hybrid network performance measurement.

Idilio Drago is a postdoctoral researcher at the Politecnico di Torino, Italy, working at the *Narus Cyber Innovation Center - Torino*. He received his Ph.D. degree from the University of Twente in 2013, and his current research interests include Internet measurements and network security.

Ramin Sadre is an assistant professor at the Distributed and Embedded Systems group of the Aalborg University, Denmark. He received a Ph.D. degree from the University of Twente for his thesis titled "Decomposition Based Analysis of Queuing Networks". His research interests include traffic modeling, the design and analytical performance evaluation of communication systems, and the design of network intrusion detection systems.

Anna Sperotto is a postdoctoral researcher at the DACS Group of the University of Twente, The Netherlands. She received a M.Sc. degree in Computer Science from the Ca'Foscari University, Venice, Italy, in 2006 and a Ph.D. degree from the University of Twente, in 2010. Her main topics of interest include intrusion detection, and traffic monitoring and modeling.

Aiko Pras is a professor in the area of Network Operations and Management at the University of Twente, The Netherlands. His research interests include network management technologies, network monitoring, measurements and security. He chairs IFIP TC6, and is coordinator of the European FP7 FLAMINGO project.